MOSEK Optimizer API for Python

Release 11.2.6

MOSEK ApS

08 October 2026

Contents

1 Introduction **1**
1.1 Why the Optimizer API for Python? . . . 2

2 Contact Information **3**

3 License Agreement **4**
3.1 **MOSEK** end-user license agreement . . . 4
3.2 Third party licenses . . . 4

4 Installation **10**
4.1 PIP . . . 10
4.2 Manual installation . . . 10
4.3 Testing the Installation . . . 11

5 Design Overview **12**
5.1 Modeling . . . 12
5.2 "Hello World!" in **MOSEK** . . . 12

6 Optimization Tutorials **14**
6.1 Linear Optimization . . . 15
6.2 From Linear to Conic Optimization . . . 21
6.3 Conic Quadratic Optimization . . . 29
6.4 Power Cone Optimization . . . 33
6.5 Conic Exponential Optimization . . . 36
6.6 Geometric Programming . . . 40
6.7 Semidefinite Optimization . . . 43
6.8 Integer Optimization . . . 53
6.9 Disjunctive constraints . . . 58
6.10 Quadratic Optimization . . . 64
6.11 Problem Modification and Reoptimization . . . 71
6.12 Parallel optimization . . . 76
6.13 Retrieving infeasibility certificates . . . 77

7 Solver Interaction Tutorials **80**
7.1 Environment and task . . . 80
7.2 Accessing the solution . . . 81
7.3 Errors and exceptions . . . 84
7.4 Input/Output . . . 86
7.5 Setting solver parameters . . . 88
7.6 Retrieving information items . . . 89
7.7 Progress and data callback . . . 90
7.8 **MOSEK** OptServer . . . 92

8 Debugging Tutorials **95**
8.1 Understanding optimizer log . . . 96
8.2 Addressing numerical issues . . . 100
8.3 Debugging infeasibility . . . 102

8.4 Python Console . . . 107

9 Advanced Numerical Tutorials **110**
9.1 Solving Linear Systems Involving the Basis Matrix . . . 110
9.2 Calling BLAS/LAPACK Routines from MOSEK . . . 117
9.3 Computing a Sparse Cholesky Factorization . . . 119

10 Technical guidelines **122**
10.1 Memory management and garbage collection . . . 122
10.2 Names . . . 122
10.3 Multithreading . . . 123
10.4 Timing . . . 123
10.5 Efficiency . . . 123
10.6 The license system . . . 125
10.7 Deployment . . . 126

11 Case Studies **127**
11.1 Portfolio Optimization . . . 127
11.2 Logistic regression . . . 148
11.3 Concurrent optimizer . . . 152

12 Problem Formulation and Solutions **156**
12.1 Linear Optimization . . . 156
12.2 Conic Optimization . . . 159
12.3 Semidefinite Optimization . . . 162
12.4 Quadratic and Quadratically Constrained Optimization . . . 164

13 Optimizers **167**
13.1 Presolve . . . 167
13.2 Linear Optimization . . . 169
13.3 Conic Optimization - Interior-point optimizer . . . 176
13.4 The Optimizer for Mixed-Integer Problems . . . 180

14 Additional features **193**
14.1 Problem Analyzer . . . 193
14.2 Automatic Repair of Infeasible Problems . . . 194
14.3 Sensitivity Analysis . . . 198

15 API Reference **206**
15.1 API Conventions . . . 206
15.2 Functions grouped by topic . . . 210
15.3 Class Env . . . 221
15.4 Class Task . . . 231
15.5 Exceptions . . . 356
15.6 Parameters grouped by topic . . . 356
15.7 Parameters (alphabetical list sorted by type) . . . 369
15.8 Response codes . . . 441
15.9 Enumerations . . . 464
15.10 Function Types . . . 496
15.11 Supported domains . . . 497
15.12 Environment variables . . . 499

16 Supported File Formats **500**
16.1 The LP File Format . . . 501
16.2 The MPS File Format . . . 505
16.3 The OPF Format . . . 517
16.4 The CBF Format . . . 528
16.5 The PTF Format . . . 545
16.6 The Task Format . . . 553

16.7 The JSON Format 553
16.8 The Solution File Format 559

17 List of examples **563**

18 Interface changes **565**
18.1 Important changes compared to version 10 565
18.2 Changes compared to version 10 565

Bibliography **571**

Symbol Index **572**

Index **590**

Chapter 1

Introduction

The **MOSEK** Optimization Suite 11.2.6 is a powerful software package capable of solving large-scale optimization problems of the following kind:

- linear,
- conic:
  - conic quadratic (also known as second-order cone),
  - involving the exponential cone,
  - involving the power cone,
  - semidefinite,
- convex quadratic and quadratically constrained,
- integer.

In order to obtain an overview of features in the **MOSEK** Optimization Suite consult the product introduction guide.

The most widespread class of optimization problems is *linear optimization problems*, where all relations are linear. The tremendous success of both applications and theory of linear optimization can be ascribed to the following factors:

- The required data are simple, i.e. just matrices and vectors.
- Convexity is guaranteed since the problem is convex by construction.
- Linear functions are trivially differentiable.
- There exist very efficient algorithms and software for solving linear problems.
- Duality properties for linear optimization are nice and simple.

Even if the linear optimization model is only an approximation to the true problem at hand, the advantages of linear optimization may outweigh the disadvantages. In some cases, however, the problem formulation is inherently nonlinear and a linear approximation is either intractable or inadequate. *Conic optimization* has proved to be a very expressive and powerful way to introduce nonlinearities, while preserving all the nice properties of linear optimization listed above.

The fundamental expression in linear optimization is a linear expression of the form

$$Ax - b \geq 0.$$

In conic optimization this is replaced with a wider class of constraints

$$Ax - b \in \mathcal{K}$$

where $\mathcal{K}$ is a *convex cone*. For example in 3 dimensions $\mathcal{K}$ may correspond to an ice cream cone. The conic optimizer in **MOSEK** supports a number of different types of cones $\mathcal{K}$, which allows a surprisingly large number of nonlinear relations to be modeled, as described in the **MOSEK** Modeling Cookbook, while preserving the nice algorithmic and theoretical properties of linear optimization.

1.1 Why the Optimizer API for Python?

The Optimizer API for Python provides an object-oriented interface to the **MOSEK** optimizers. This object oriented design is common to Java, Python and .NET and is based on a thin class-based interface to the native C optimizer API. The overhead introduced by this mapping is minimal.

The Optimizer API for Python can be used with any application running on recent Python 3 interpreters. It consists of a single `mosek` package which can be used in Python scripts and interactive shells making it suited for fast prototyping and inspection of models.

The Optimizer API for Python provides access to:

- Linear Optimization (LO)
- Conic Quadratic (Second-Order Cone) Optimization (CQO, SOCO)
- Power Cone Optimization
- Conic Exponential Optimization (CEO)
- Convex Quadratic and Quadratically Constrained Optimization (QO, QCQO)
- Semidefinite Optimization (SDO)
- Mixed-Integer Optimization (MIO) including Disjunctive Constraints (DJC)

as well as to additional functions for

- problem analysis,
- sensitivity analysis,
- infeasibility diagnostics,
- BLAS/LAPACK linear algebra routines.

Chapter 2

Contact Information

Phone	+45 7174 9373	Office
	+45 7174 5700	Sales
Website	mosek.com	
Email		
	sales@mosek.com	Sales, pricing, and licensing
	support@mosek.com	Technical support, questions and bug reports
	info@mosek.com	Everything else.
Mailing Address		
	MOSEK ApS	
	Fruebjergvej 3	
	Symbion Science Park, Box 16	
	2100 Copenhagen O	
	Denmark	

You can get in touch with **MOSEK** using popular social media as well:

Blogger	https://blog.mosek.com/
Google Group	https://groups.google.com/forum/#!forum/mosek
Linkedin	https://www.linkedin.com/company/mosek-aps
Youtube	https://www.youtube.com/channel/UCvIyectEVLP31NXeD5mIbEw

Chapter 3

License Agreement

3.1 MOSEK end-user license agreement

Before using the **MOSEK** software, please read the license agreement available in the distribution at `<MSKHOME>/mosek/11.2/mosek-eula.pdf` or on the **MOSEK** website https://mosek.com/products/license-agreement. By using **MOSEK** you agree to the terms of that license agreement.

3.2 Third party licenses

MOSEK uses some third-party open-source libraries. Their license details follow.

zlib

MOSEK uses the *zlib* library obtained from the zlib website. The license agreement for *zlib* is shown in Listing 3.1.

Listing 3.1: *zlib* license.

```
zlib.h -- interface of the 'zlib' general purpose compression library
version 1.3.2, February 17th, 2026

Copyright (C) 1995-2026 Jean-loup Gailly and Mark Adler

This software is provided 'as-is', without any express or implied
warranty.  In no event will the authors be held liable for any damages
arising from the use of this software.

Permission is granted to anyone to use this software for any purpose,
including commercial applications, and to alter it and redistribute it
freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not
   claim that you wrote the original software. If you use this software
   in a product, an acknowledgment in the product documentation would be
   appreciated but is not required.
2. Altered source versions must be plainly marked as such, and must not be
   misrepresented as being the original software.
3. This notice may not be removed or altered from any source distribution.

Jean-loup Gailly        Mark Adler
jloup@gzip.org          madler@alumni.caltech.edu
```

fplib

MOSEK uses the floating point formatting library developed by David M. Gay obtained from the netlib website. The license agreement for *fplib* is shown in Listing 3.2.

Listing 3.2: *fplib* license.

```
/****************************************************************
 *
 * The author of this software is David M. Gay.
 *
 * Copyright (c) 1991, 2000, 2001 by Lucent Technologies.
 *
 * Permission to use, copy, modify, and distribute this software for any
 * purpose without fee is hereby granted, provided that this entire notice
 * is included in all copies of any software which is or includes a copy
 * or modification of this software and in all copies of the supporting
 * documentation for such software.
 *
 * THIS SOFTWARE IS BEING PROVIDED "AS IS", WITHOUT ANY EXPRESS OR IMPLIED
 * WARRANTY.  IN PARTICULAR, NEITHER THE AUTHOR NOR LUCENT MAKES ANY
 * REPRESENTATION OR WARRANTY OF ANY KIND CONCERNING THE MERCHANTABILITY
 * OF THIS SOFTWARE OR ITS FITNESS FOR ANY PARTICULAR PURPOSE.
 *
 ***************************************************************/
```

{fmt}

MOSEK uses the formatting library *{fmt}* developed by Victor Zverovich obtained form github/fmt and distributed under the MIT license. The license agreement fot *{fmt}* is shown in Listing 3.3.

Listing 3.3: *{fmt}* license.

```
Copyright (c) 2012 - present, Victor Zverovich

Permission is hereby granted, free of charge, to any person obtaining
a copy of this software and associated documentation files (the "Software"),
to deal in the Software without restriction, including without limitation
the rights to use, copy, modify, merge, publish, distribute, sublicense,
and/or sell copies of the Software, and to permit persons to whom the Software
is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included
in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR
A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER
IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

Zstandard

MOSEK uses the *Zstandard* library developed by Facebook obtained from github/zstd. The license agreement for *Zstandard* is shown in Listing 3.4.

Listing 3.4: *Zstandard* license.

```
BSD License

For Zstandard software

Copyright (c) 2016-present, Facebook, Inc. All rights reserved.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this
  list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice,
  this list of conditions and the following disclaimer in the documentation
  and/or other materials provided with the distribution.

* Neither the name Facebook nor the names of its contributors may be used to
  endorse or promote products derived from this software without specific
  prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

OpenSSL

MOSEK uses the LibReSSL library, which is build on *OpenSSL*. OpenSSL is included under the OpenSSL license, Listing 3.5, and the LibReSSL additions are licensed under the ISC license, Listing 3.6.

Listing 3.5: *OpenSSL* license

```
====================================================================
Copyright (c) 1998-2011 The OpenSSL Project.  All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in
```

(continues on next page)

(continued from previous page)

```
   the documentation and/or other materials provided with the
   distribution.

3. All advertising materials mentioning features or use of this
   software must display the following acknowledgment:
   "This product includes software developed by the OpenSSL Project
   for use in the OpenSSL Toolkit. (http://www.openssl.org/)"

4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to
   endorse or promote products derived from this software without
   prior written permission. For written permission, please contact
   openssl-core@openssl.org.

5. Products derived from this software may not be called "OpenSSL"
   nor may "OpenSSL" appear in their names without prior written
   permission of the OpenSSL Project.

6. Redistributions of any form whatsoever must retain the following
   acknowledgment:
   "This product includes software developed by the OpenSSL Project
   for use in the OpenSSL Toolkit (http://www.openssl.org/)"

THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY
EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE OpenSSL PROJECT OR
ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
OF THE POSSIBILITY OF SUCH DAMAGE.
====================================================================

This product includes cryptographic software written by Eric Young
(eay@cryptsoft.com).  This product includes software written by Tim
Hudson (tjh@cryptsoft.com).
```

Listing 3.6: *ISC* license

```
Copyright (C) 1994-2017 Free Software Foundation, Inc.
Copyright (c) 2014 Jeremie Courreges-Anglas <jca@openbsd.org>
Copyright (c) 2014-2015 Joel Sing <jsing@openbsd.org>
Copyright (c) 2014 Ted Unangst <tedu@openbsd.org>
Copyright (c) 2015-2016 Bob Beck <beck@openbsd.org>
Copyright (c) 2015 Marko Kreen <markokr@gmail.com>
Copyright (c) 2015 Reyk Floeter <reyk@openbsd.org>
Copyright (c) 2016 Tobias Pape <tobias@netshed.de>

Permission to use, copy, modify, and/or distribute this software for
any purpose with or without fee is hereby granted, provided that the
above copyright notice and this permission notice appear in all
copies.
```

(continues on next page)

(continued from previous page)

```
THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL
WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE
AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL
DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR
PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER
TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
PERFORMANCE OF THIS SOFTWARE.
```

mimalloc

MOSEK uses the *mimalloc* memory allocator library from github/mimalloc. The license agreement for *mimalloc* is shown in Listing 3.7.

Listing 3.7: *mimalloc* license.

```
MIT License

Copyright (c) 2019 Microsoft Corporation, Daan Leijen

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

BLASFEO

MOSEK uses the *BLASFEO* linear algebra library developed by Gianluca Frison, obtained from github/blasfeo. The license agreement for *BLASFEO* is shown in Listing 3.8.

Listing 3.8: *blasfeo* license.

```
BLASFEO -- BLAS For Embedded Optimization.
Copyright (C) 2019 by Gianluca Frison.
Developed at IMTEK (University of Freiburg) under the supervision of Moritz Diehl.
All rights reserved.

The 2-Clause BSD License

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this
```

(continues on next page)

(continued from previous page)

```
   list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice,
   this list of conditions and the following disclaimer in the documentation
   and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR
ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

oneTBB

MOSEK uses the *oneTBB* parallelization library which is part of *oneAPI* developed by Intel, obtained from github/oneTBB, licensed under the Apache License 2.0. The license agreement for *oneTBB* can be found in https://github.com/oneapi-src/oneTBB/blob/master/LICENSE.txt .

Chapter 4

Installation

In this section we discuss how to install and setup the **MOSEK** Optimizer API for Python.

Important: Before running this **MOSEK** interface please make sure that you:

- Installed **MOSEK** correctly. Some operating systems require extra steps. See the Installation guide for instructions and common troubleshooting tips.
- Set up a license. See the Licensing guide for instructions.

Compatibility

The Optimizer API for Python requires Python 3. The supported versions of Python are 3.9-3.14.

4.1 PIP

The recommended way to install **MOSEK** Optimization Suite is via PIP:

```
pip install mosek
```

(alternatively with `--user` for a user installation; all standard `pip` options apply).
If you installed the **MOSEK** package with PIP, no additional setup is required.

4.2 Manual installation

Locating files in the MOSEK Optimization Suite

The relevant files of the Optimizer API for Python are organized as reported in Table 4.1.

Table 4.1: Relevant files for the Optimizer API for Python.

Relative Path	Description	Label
`<MSKHOME>/mosek/11.2/tools/platform/<PLATFORM>/python/3`	Python 3 install	`<PYTHON3DIR>`
`<MSKHOME>/mosek/11.2/tools/examples/python`	Examples	`<EXDIR>`
`<MSKHOME>/mosek/11.2/tools/examples/data`	Additional data	`<MISCDIR>`

where

- `<MSKHOME>` is the folder in which the **MOSEK** Optimization Suite has been installed,
- `<PLATFORM>` is the actual platform among those supported by **MOSEK**, i.e. `win64x86`, `linux64x86`, `linuxaarch64` or `osxaarch64`.

Manual install and setting up paths

You can install **MOSEK** for Python manually from the **MOSEK** distribution using `pip`. At least version `21.3` of `pip` is required, as it allows for in-tree builds. Go to the folder `<MSKHOME>/mosek/11.2/tools/platform/<PLATFORM>/python/3` and run

```
pip install .
```

All standard options for `pip` can be applied according to the user's needs.

4.3 Testing the Installation

First of all, to check that the Optimizer API for Python was properly installed, start Python and try

```
import mosek
```

The installation can further be tested by running some of the enclosed examples. Open a terminal, change folder to `<EXDIR>` and use Python to run a selected example, for instance:

```
python lo1.py
```

Chapter 5

Design Overview

5.1 Modeling

Optimizer API for Python is an interface for specifying optimization problems directly in matrix form. It means that an optimization problem such as:

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax \leq b, \\ & x \in \mathcal{K} \end{array}$$

is specified by describing the matrix A, vectors b, c and a list of cones $\mathcal{K}$ directly.

The main characteristics of this interface are:

- **Simplicity**: once the problem data is assembled in matrix form, it is straightforward to input it into the optimizer.
- **Exploiting sparsity**: data is entered in sparse format, enabling huge, sparse problems to be defined and solved efficiently.
- **Efficiency**: the Optimizer API incurs almost no overhead between the user's representation of the problem and **MOSEK**'s internal one.

Optimizer API for Python does not aid with modeling. It is the user's responsibility to express the problem in **MOSEK**'s standard form, introducing, if necessary, auxiliary variables and constraints. See Sec. 12 for the precise formulations of problems **MOSEK** solves.

5.2 "Hello World!" in MOSEK

Here we present the most basic workflow pattern when using Optimizer API for Python.

Creating an environment and task

Optionally, an interaction with **MOSEK** using Optimizer API for Python can begin by creating a **MOSEK environment**. It coordinates the access to **MOSEK** from the current process.

In most cases the user does not interact directly with the environment, except for creating optimization **tasks**, which contain actual problem specifications and where optimization takes place. In this case the user can directly create tasks without invoking an environment, as we do here.

Defining tasks

After a task is created, the input data can be specified. An optimization problem consists of several components; objective, objective sense, constraints, variable bounds etc. See Sec. 6 for basic tutorials on how to specify and solve various types of optimization problems.

Retrieving the solutions

When the model is set up, the optimizer is invoked with the call to *Task.optimize*. When the optimization is over, the user can check the results and retrieve numerical values. See further details in Sec. 7.

We refer also to Sec. 7 for information about more advanced mechanisms of interacting with the solver.

Source code example

Below is the most basic code sample that defines and solves a trivial optimization problem

$$\begin{array}{ll} \text{minimize} & x \\ \text{subject to} & 2.0 \leq x \leq 3.0. \end{array}$$

For simplicity the example does not contain any error or status checks.

Listing 5.1: "Hello World!" in MOSEK

```
##
#  Copyright: Copyright (c) MOSEK ApS, Denmark. All rights reserved.
#
#  File:      helloworld.py
#
#  The most basic example of how to get started with MOSEK.

from mosek import *

with Task() as task:                          # Create Task
  task.appendvars(1)                          # 1 variable x
  task.putcj(0, 1.0)                          # c_0 = 1.0
  task.putvarbound(0, boundkey.ra, 2.0, 3.0)  # 2.0 <= x <= 3.0
  task.putobjsense(objsense.minimize)         # minimize

  task.optimize()                            # Optimize

  x = task.getxx(soltype.itr)                # Get solution
  print("Solution x = {}".format(x[0]))      # Print solution
```

Chapter 6

Optimization Tutorials

In this section we demonstrate how to set up basic types of optimization problems. Each short tutorial contains a working example of formulating problems, defining variables and constraints and retrieving solutions.

- **Model setup and linear optimization tutorial (LO)**
  - Sec. 6.1. Linear optimization tutorial, *recommended first reading for all users.* Apart from setting up a linear problem it also demonstrates how to work with an optimizer task: initialize it, add variables and constraints and retrieve the solution.
- **Conic optimization tutorials (CO)**
  - Sec. 6.2. A step by step introduction to programming with affine conic constraints (ACC). Explains all the steps required to input a conic problem. *Recommended first reading for users of the conic optimizer.*

  Further basic examples demonstrating various types of conic constraints:
  - Sec. 6.3. A basic example with a quadratic cone (CQO).
  - Sec. 6.4. A basic example with a power cone.
  - Sec. 6.5. A basic example with a exponential cone (CEO).
  - Sec. 6.6. A basic tutorial of geometric programming (GP).
- **Semidefinite optimization tutorial (SDO)**
  - Sec. 6.7. Examples showing how to solve semidefinite optimization problems with one or more semidefinite variables.
- **Mixed-integer optimization tutorials (MIO)**
  - Sec. 6.8. Shows how to declare integer variables for linear and conic problems and how to set an initial solution.
  - Sec. 6.9. Demonstrates how to create a problem with disjunctive constraints (DJC).
- **Quadratic optimization tutorial (QO, QCQO)**
  - Sec. 6.10. Examples showing how to solve a quadratic or quadratically constrained problem.
- **Reoptimization tutorials**
  - Sec. 6.11. Various techniques for modifying and reoptimizing a problem.
- **Parallel optimization tutorial**
  - Sec. 6.12. Shows how to optimize tasks in parallel.
- **Infeasibility certificates**
  - Sec. 6.13. Shows how to retrieve and analyze a primal infeasibility certificate for continuous problems.

6.1 Linear Optimization

The simplest optimization problem is a purely linear problem. A *linear optimization problem* (see also Sec. 12.1) is a problem of the following form:

Minimize or maximize the objective function

$$\sum_{j=0}^{n-1} c_j x_j + c^f$$

subject to the linear constraints

$$l_k^c \leq \sum_{j=0}^{n-1} a_{kj} x_j \leq u_k^c, \quad k = 0, \ldots, m-1,$$

and the bounds

$$l_j^x \leq x_j \leq u_j^x, \quad j = 0, \ldots, n-1.$$

The problem description consists of the following elements:

- m and n — the number of constraints and variables, respectively,
- x — the variable vector of length n,
- c — the coefficient vector of length n

$$c = \begin{bmatrix} c_0 \\ \vdots \\ c_{n-1} \end{bmatrix},$$

- c^f — fixed term in the objective,
- A — an $m \times n$ matrix of coefficients

$$A = \begin{bmatrix} a_{0,0} & \cdots & a_{0,(n-1)} \\ \vdots & \ldots & \vdots \\ a_{(m-1),0} & \cdots & a_{(m-1),(n-1)} \end{bmatrix},$$

- l^c and u^c — the lower and upper bounds on constraints,
- l^x and u^x — the lower and upper bounds on variables.

Please note that we are using 0 as the first index: x_0 is the first element in variable vector x.

6.1.1 Example LO1

The following is an example of a small linear optimization problem:

$$\begin{array}{lrcrcrcrcl} \text{maximize} & 3x_0 & + & 1x_1 & + & 5x_2 & + & 1x_3 & & \\ \text{subject to} & 3x_0 & + & 1x_1 & + & 2x_2 & & & = & 30, \\ & 2x_0 & + & 1x_1 & + & 3x_2 & + & 1x_3 & \geq & 15, \\ & & & 2x_1 & & & + & 3x_3 & \leq & 25, \end{array} \tag{6.1}$$

under the bounds

$$\begin{array}{rcrcl} 0 & \leq & x_0 & \leq & \infty, \\ 0 & \leq & x_1 & \leq & 10, \\ 0 & \leq & x_2 & \leq & \infty, \\ 0 & \leq & x_3 & \leq & \infty. \end{array}$$

Solving the problem

To solve the problem above we go through the following steps:

1. (Optionally) Creating an environment.
2. Creating an optimization task.
3. Loading a problem into the task object.
4. Optimization.
5. Extracting the solution.

Below we explain each of these steps.

Creating an environment.

The user can start by creating a **MOSEK** environment, but it is not necessary if the user does not need access to other functionalities, license management, additional routines, etc. Therefore in this tutorial we don't create an explicit environment.

Creating an optimization task.

We create an empty task object. A task object represents all the data (inputs, outputs, parameters, information items etc.) associated with one optimization problem.

```
    # Create a task object
    with mosek.Task() as task:
        # Attach a log stream printer to the task
        task.set_Stream(mosek.streamtype.log, streamprinter)
```

We also connect a call-back function to the task log stream. Messages related to the task are passed to the call-back function. In this case the stream call-back function writes its messages to the standard output stream. See Sec. 7.4.

Loading a problem into the task object.

Before any problem data can be set, variables and constraints must be added to the problem via calls to the functions *Task.appendcons* and *Task.appendvars*.

```
        # Append 'numcon' empty constraints.
        # The constraints will initially have no bounds.
        task.appendcons(numcon)

        # Append 'numvar' variables.
        # The variables will initially be fixed at zero (x=0).
        task.appendvars(numvar)
```

New variables can now be referenced from other functions with indexes in $0, \ldots,$ `numvar` $- 1$ and new constraints can be referenced with indexes in $0, \ldots,$ `numcon` $- 1$. More variables and/or constraints can be appended later as needed, these will be assigned indexes from `numvar`/`numcon` and up. Optionally one can add names.

Setting the objective.

Next step is to set the problem data. We first set the objective coefficients $c_j = \texttt{c[j]}$. This can be done with functions such as *Task.putcj* or *Task.putclist*.

```
task.putcj(j, c[j])
```

Setting bounds on variables

For every variable we need to specify a bound key and two bounds according to Table 6.1.

Table 6.1: Bound keys as defined in the enum `boundkey`.

Bound key	Type of bound	Lower bound	Upper bound
boundkey.fx	$u_j = l_j$	Finite	Identical to the lower bound
boundkey.fr	Free	$-\infty$	$+\infty$
boundkey.lo	$l_j \leq \cdots$	Finite	$+\infty$
boundkey.ra	$l_j \leq \cdots \leq u_j$	Finite	Finite
boundkey.up	$\cdots \leq u_j$	$-\infty$	Finite

For instance `bkx[0]= boundkey.lo` means that $x_0 \geq l_0^x$. Finally, the numerical values of the bounds on variables are given by

$$l_j^x = \texttt{blx[j]}$$

and

$$u_j^x = \texttt{bux[j]}.$$

Let us assume we have the bounds on variables stored in the arrays

```
# Bound keys for variables
bkx = [mosek.boundkey.lo,
       mosek.boundkey.ra,
       mosek.boundkey.lo,
       mosek.boundkey.lo]

# Bound values for variables
blx = [0.0, 0.0, 0.0, 0.0]
bux = [+inf, 10.0, +inf, +inf]
```

Then we can set them using various functions such *Task.putvarbound*, *Task.putvarboundslice*, *Task.putvarboundlist*, depending on what is most convenient in the given context. For instance:

```
# Set the bounds on variable j
# blx[j] <= x_j <= bux[j]
task.putvarbound(j, bkx[j], blx[j], bux[j])
```

Defining the linear constraint matrix.

Recall that in our example the A matrix is given by

$$A = \begin{bmatrix} 3 & 1 & 2 & 0 \\ 2 & 1 & 3 & 1 \\ 0 & 2 & 0 & 3 \end{bmatrix}.$$

This matrix is stored in sparse format:

```
        asub = [[0, 1],
                [0, 1, 2],
                [0, 1],
                [1, 2]]
        aval = [[3.0, 2.0],
                [1.0, 1.0, 2.0],
                [2.0, 3.0],
                [1.0, 3.0]]
```

The array `aval[j]` contains the non-zero values of column j and `asub[j]` contains the row indices of these non-zeros.

We now input the linear constraint matrix into the task. This can be done in many alternative ways, row-wise, column-wise or element by element in various orders. See functions such as *Task.putarow*, *Task.putarowlist*, *Task.putaijlist*, *Task.putacol* and similar.

```
            task.putacol(j,                 # Variable (column) index.
                         asub[j],           # Row index of non-zeros in column j.
                         aval[j])           # Non-zero Values of column j.
```

Setting bounds on constraints

Finally, the bounds on each constraint are set similarly to the variable bounds, using the bound keys as in Table 6.1. This can be done with one of the many functions *Task.putconbound*, *Task.putconboundslice*, *Task.putconboundlist*, depending on the situation.

```
        # Set the bounds on constraints.
         # blc[i] <= constraint_i <= buc[i]
        for i in range(numcon):
            task.putconbound(i, bkc[i], blc[i], buc[i])
```

Optimization

After the problem is set-up the task can be optimized by calling the function *Task.optimize*.

```
        task.optimize()
```

Extracting the solution.

After optimizing the status of the solution is examined with a call to *Task.getsolsta*.

```
        # Get status information about the solution
        solsta = task.getsolsta(mosek.soltype.bas)
```

If the solution status is reported as *solsta.optimal* the solution is extracted:

```
            xx = task.getxx(mosek.soltype.bas)
```

The *Task.getxx* function obtains the solution. **MOSEK** may compute several solutions depending on the optimizer employed. In this example the *basic solution* is requested by setting the first argument to *soltype.bas*. For details about fetching solutions see Sec. 7.2.

Catching exceptions

We catch any exceptions thrown by **MOSEK** in the lines:

```
except mosek.Error as e:
    print("ERROR: %s" % str(e.errno))
    if e.msg is not None:
        print("\t%s" % e.msg)
        sys.exit(1)
```

The types of exceptions that **MOSEK** can throw can be seen in Sec. 15.5. See also Sec. 7.3.

Source code

The complete source code `lo1.py` of this example appears below. See also `lo2.py` for a version where the A matrix is entered row-wise.

Listing 6.1: Linear optimization example.

```
import sys
import mosek

# Since the value of infinity is ignored, we define it solely
# for symbolic purposes
inf = 0.0

# Define a stream printer to grab output from MOSEK
def streamprinter(text):
    sys.stdout.write(text)
    sys.stdout.flush()


def main():
    # Create a task object
    with mosek.Task() as task:
        # Attach a log stream printer to the task
        task.set_Stream(mosek.streamtype.log, streamprinter)

        # Bound keys for constraints
        bkc = [mosek.boundkey.fx,
               mosek.boundkey.lo,
               mosek.boundkey.up]

        # Bound values for constraints
        blc = [30.0, 15.0, -inf]
        buc = [30.0, +inf, 25.0]

        # Bound keys for variables
        bkx = [mosek.boundkey.lo,
               mosek.boundkey.ra,
               mosek.boundkey.lo,
               mosek.boundkey.lo]

        # Bound values for variables
        blx = [0.0, 0.0, 0.0, 0.0]
        bux = [+inf, 10.0, +inf, +inf]

        # Objective coefficients
        c = [3.0, 1.0, 5.0, 1.0]
```

(continues on next page)

(continued from previous page)

```
        # Below is the sparse representation of the A
        # matrix stored by column.
        asub = [[0, 1],
                [0, 1, 2],
                [0, 1],
                [1, 2]]
        aval = [[3.0, 2.0],
                [1.0, 1.0, 2.0],
                [2.0, 3.0],
                [1.0, 3.0]]

        numvar = len(bkx)
        numcon = len(bkc)

        # Append 'numcon' empty constraints.
        # The constraints will initially have no bounds.
        task.appendcons(numcon)

        # Append 'numvar' variables.
        # The variables will initially be fixed at zero (x=0).
        task.appendvars(numvar)

        for j in range(numvar):
            # Set the linear term c_j in the objective.
            task.putcj(j, c[j])

            # Set the bounds on variable j
            # blx[j] <= x_j <= bux[j]
            task.putvarbound(j, bkx[j], blx[j], bux[j])

            # Input column j of A
            task.putacol(j,                 # Variable (column) index.
                         asub[j],           # Row index of non-zeros in column j.
                         aval[j])           # Non-zero Values of column j.

        # Set the bounds on constraints.
         # blc[i] <= constraint_i <= buc[i]
        for i in range(numcon):
            task.putconbound(i, bkc[i], blc[i], buc[i])

        # Input the objective sense (minimize/maximize)
        task.putobjsense(mosek.objsense.maximize)

        # Solve the problem
        task.optimize()
        # Print a summary containing information
        # about the solution for debugging purposes
        task.solutionsummary(mosek.streamtype.msg)

        # Get status information about the solution
        solsta = task.getsolsta(mosek.soltype.bas)

        if (solsta == mosek.solsta.optimal):
            xx = task.getxx(mosek.soltype.bas)
```

(continues on next page)

(continued from previous page)

```
            print("Optimal solution: ")
            for i in range(numvar):
                print("x[" + str(i) + "]=" + str(xx[i]))
        elif (solsta == mosek.solsta.dual_infeas_cer or
              solsta == mosek.solsta.prim_infeas_cer):
            print("Primal or dual infeasibility certificate found.\n")
        elif solsta == mosek.solsta.unknown:
            print("Unknown solution status")
        else:
            print("Other solution status")


# call the main function
try:
    main()
except mosek.Error as e:
    print("ERROR: %s" % str(e.errno))
    if e.msg is not None:
        print("\t%s" % e.msg)
        sys.exit(1)
except:
    import traceback
    traceback.print_exc()
    sys.exit(1)
```

6.2 From Linear to Conic Optimization

In Sec. 6.1 we demonstrated setting up the linear part of an optimization problem, that is the objective, linear bounds, linear equalities and inequalities. In this tutorial we show how to define conic constraints. We recommend going through this general conic tutorial before proceeding to examples with specific cone types.

MOSEK accepts conic constraints in the form

$$Fx + g \in \mathcal{D}$$

where

- $x \in \mathbb{R}^{n\text{‘}}$ is the optimization variable,
- $D \subseteq \mathbb{R}^k$ is a **conic domain** of some dimension k, representing *one of the cone types supported by MOSEK*,
- $F \in \mathbb{R}^{k \times n}$ and $g \in \mathbb{R}^k$ are data which constitute the sequence of k **affine expressions** appearing in the rows of $Fx + g$.

Constraints of this form will be called **affine conic constraints**, or **ACC** for short. Therefore in this section we show how to set up a problem of the form

$$\begin{array}{lccccc} \text{minimize} & & & c^T x + c^f & & \\ \text{subject to} & l^c & \leq & Ax & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x, \\ & & & Fx + g & \in & \mathcal{D}_1 \times \cdots \times \mathcal{D}_p, \end{array}$$

with some number p of affine conic constraints.

Note that conic constraints are a natural generalization of linear constraints to the general nonlinear case. For example, a typical linear constraint of the form

$$Ax + b \geq 0$$

can be also written as membership in the cone of nonnegative real numbers:

$$Ax + b \in \mathbb{R}^d_{\geq 0},$$

and that naturally generalizes to

$$Fx + g \in \mathcal{D}$$

for more complicated domains $\mathcal{D}$ from Sec. 15.11 of which $\mathcal{D} = \mathbb{R}^d_{\geq 0}$ is a special case.

6.2.1 Running example

In this tutorial we will consider a sample problem of the form

$$\begin{array}{ll} \text{maximize} & c^T x \\ \text{subject to} & \sum_i x_i = 1, \\ & \gamma \geq \|Gx + h\|_2, \end{array} \tag{6.2}$$

where $x \in \mathbb{R}^n$ is the optimization variable and $G \in \mathbb{R}^{k \times n}$, $h \in \mathbb{R}^k$, $c \in \mathbb{R}^n$ and $\gamma \in \mathbb{R}$. We will use the following sample data:

$$n = 3, \quad k = 2, \quad x \in \mathbb{R}^3, \quad c = [2, 3, -1]^T, \quad \gamma = 0.03, \quad G = \begin{bmatrix} 1.5 & 0.1 & 0 \\ 0.3 & 0 & 2.1 \end{bmatrix}, \quad h = \begin{bmatrix} 0 \\ 0.1 \end{bmatrix}.$$

To be explicit, the problem we are going to solve is therefore:

$$\begin{array}{ll} \text{maximize} & 2x_0 + 3x_1 - x_2 \\ \text{subject to} & x_0 + x_1 + x_2 = 1, \\ & 0.03 \geq \sqrt{(1.5x_0 + 0.1x_1)^2 + (0.3x_0 + 2.1x_2 + 0.1)^2}. \end{array} \tag{6.3}$$

Consulting the *definition of a quadratic cone* $\mathcal{Q}$ we see that the conic form of this problem is:

$$\begin{array}{ll} \text{maximize} & 2x_0 + 3x_1 - x_2 \\ \text{subject to} & x_0 + x_1 + x_2 = 1, \\ & (0.03,\ 1.5x_0 + 0.1x_1,\ 0.3x_0 + 2.1x_2 + 0.1) \in \mathcal{Q}^3. \end{array} \tag{6.4}$$

The conic constraint has an affine conic representation $Fx + g \in \mathcal{D}$ as follows:

$$\begin{bmatrix} 0 & 0 & 0 \\ 1.5 & 0.1 & 0 \\ 0.3 & 0 & 2.1 \end{bmatrix} x + \begin{bmatrix} 0.03 \\ 0 \\ 0.1 \end{bmatrix} \in \mathcal{Q}^3. \tag{6.5}$$

Of course by the same logic in the general case the conic form of the problem (6.2) would be

$$\begin{array}{ll} \text{maximize} & c^T x \\ \text{subject to} & \sum_i x_i = 1, \\ & (\gamma, Gx + h) \in \mathcal{Q}^{k+1} \end{array} \tag{6.6}$$

and the ACC representation of the constraint $(\gamma, Gx + h) \in \mathcal{Q}^{k+1}$ would be

$$\begin{bmatrix} 0 \\ G \end{bmatrix} x + \begin{bmatrix} \gamma \\ h \end{bmatrix} \in \mathcal{Q}^{k+1}.$$

Now we show how to add the ACC (6.5). This involves three steps:

- storing the affine expressions which appear in the constraint,
- creating a domain, and
- combining the two into an ACC.

6.2.2 Step 1: add affine expressions

To store affine expressions (**AFE** for short) **MOSEK** provides a matrix $\mathbf{F}$ and a vector $\mathbf{g}$ with the understanding that every row of

$$\mathbf{F}x + \mathbf{g}$$

defines one affine expression. The API functions with infix `afe` are used to operate on $\mathbf{F}$ and $\mathbf{g}$, add rows, add columns, set individual elements, set blocks etc. similarly to the methods for operating on the A matrix of linear constraints. The storage matrix $\mathbf{F}$ is a sparse matrix, therefore only nonzero elements have to be explicitly added.

Remark: the storage $\mathbf{F}, \mathbf{g}$ may, but does not have to be, equal to the pair F, g appearing in the expression $Fx + g$. It is possible to store the AFEs in different order than the order they will be used in F, g, as well as store some expressions only once if they appear multiple times in $Fx + g$. In this first turorial, however, we will for simplicity store all expressions in the same order we will later use them, so that $(\mathbf{F}, \mathbf{g}) = (F, g)$.

In our example we create only one conic constraint (6.5) with three (in general $k+1$) affine expressions

$$\begin{aligned} &0.03, \\ &1.5x_0 + 0.1x_1, \\ &0.3x_0 + 2.1x_2 + 0.1. \end{aligned}$$

Given the previous remark, we initialize the AFE storage as:

$$\mathbf{F} = \begin{bmatrix} 0 & 0 & 0 \\ 1.5 & 0.1 & 0 \\ 0.3 & 0 & 2.1 \end{bmatrix}, \quad \mathbf{g} = \begin{bmatrix} 0.03 \\ 0 \\ 0.1 \end{bmatrix}. \tag{6.7}$$

Initially $\mathbf{F}$ and $\mathbf{g}$ are empty (have 0 rows). We construct them as follows. First, we append a number of empty rows:

```
# Append empty AFE rows for affine expression storage
task.appendafes(k + 1)
```

We now have $\mathbf{F}$ and $\mathbf{g}$ with 3 rows of zeros and we fill them up to obtain (6.7).

```
# G matrix in sparse form
Gsubi = [0, 0, 1, 1]
Gsubj = [0, 1, 0, 2]
Gval  = [1.5, 0.1, 0.3, 2.1]
# Other data
h     = [0, 0.1]
gamma = 0.03

# Construct F matrix in sparse form
Fsubi = [i + 1 for i in Gsubi]   # G will be placed from row number 1 in F
Fsubj = Gsubj
Fval  = Gval

# Fill in F storage
task.putafefentrylist(Fsubi, Fsubj, Fval)

# Fill in g storage
task.putafeg(0, gamma)
task.putafegslice(1, k+1, h)
```

We have now created the matrices from (6.7). Note that at this point we have *not defined any ACC yet*. All we did was define some affine expressions and place them in a generic AFE storage facility to be used later.

6.2.3 Step 2: create a domain

Next, we create the domain to which the ACC belongs. Domains are created with functions with infix `domain`. In the case of (6.5) we need a quadratic cone domain of dimension 3 (in general $k+1$), which we create with:

```
    # Define a conic quadratic domain
    quadDom = task.appendquadraticconedomain(k + 1)
```

The function returns a domain index, which is just the position in the list of all domains (potentially) created for the problem. At this point the domain is just stored in the list of domains, but not yet used for anything.

6.2.4 Step 3: create the actual constraint

We are now in position to create the affine conic constraint. ACCs are created with functions with infix `acc`. The most basic variant, *Task.appendacc* will append an affine conic constraint based on the following data:

- the list `afeidx` of indices of AFEs to be used in the constraint. These are the row numbers in $\mathbf{F}, \mathbf{g}$ which contain the required affine expressions.
- the index `domidx` of the domain to which the constraint belongs.

Note that number of AFEs used in `afeidx` must match the dimension of the domain.

In case of (6.5) we have already arranged $\mathbf{F}, \mathbf{g}$ in such a way that their (only) three rows contain the three affine expressions we need (in the correct order), and we already defined the quadratic cone domain of matching dimension 3. The ACC is now constructed with the following call:

```
    # Create the ACC
    task.appendacc(quadDom,     # Domain index
                   range(k+1), # Indices of AFE rows [0,...,k]
                   None)        # Ignored
```

This completes the setup of the affine conic constraint.

6.2.5 Example ACC1

We refer to Sec. 6.1 for instructions how to set up the objective and linear constraint $x_0 + x_1 + x_2 = 1$. All else that remains is to set up the **MOSEK** environment, task, add variables, call the solver with *Task.optimize* and retrieve the solution with *Task.getxx*. Since our problem contains a nonlinear constraint we fetch the interior-point solution. The full code solving problem (6.3) is shown below.

Listing 6.2: Full code of example ACC1.

```
import sys, mosek

# Define a stream printer to grab output from MOSEK
def streamprinter(text):
    sys.stdout.write(text)
    sys.stdout.flush()

# Define problem data
n, k = 3, 2
# Only a symbolic constant
inf = 0.0

# Create a task
with mosek.Task() as task:
    # Attach a printer to the task
    task.set_Stream(mosek.streamtype.log, streamprinter)
```

(continues on next page)

(continued from previous page)

```
    # Create n free variables
    task.appendvars(n)
    task.putvarboundsliceconst(0, n, mosek.boundkey.fr, -inf, inf)

    # Set up the objective
    c = [2, 3, -1]
    task.putobjsense(mosek.objsense.maximize)
    task.putclist(range(n), c)

    # One linear constraint - sum(x) = 1
    task.appendcons(1)
    task.putarow(0, range(n), [1]*n)
    task.putconbound(0, mosek.boundkey.fx, 1.0, 1.0)

    # Append empty AFE rows for affine expression storage
    task.appendafes(k + 1)

    # G matrix in sparse form
    Gsubi = [0, 0, 1, 1]
    Gsubj = [0, 1, 0, 2]
    Gval  = [1.5, 0.1, 0.3, 2.1]
    # Other data
    h     = [0, 0.1]
    gamma = 0.03

    # Construct F matrix in sparse form
    Fsubi = [i + 1 for i in Gsubi]    # G will be placed from row number 1 in F
    Fsubj = Gsubj
    Fval  = Gval

    # Fill in F storage
    task.putafefentrylist(Fsubi, Fsubj, Fval)

    # Fill in g storage
    task.putafeg(0, gamma)
    task.putafegslice(1, k+1, h)

    # Define a conic quadratic domain
    quadDom = task.appendquadraticconedomain(k + 1)

    # Create the ACC
    task.appendacc(quadDom,     # Domain index
                   range(k+1), # Indices of AFE rows [0,...,k]
                   None)        # Ignored

    task.writedata("acc1.ptf")
    # Solve and retrieve solution
    task.optimize()
    xx = task.getxx(mosek.soltype.itr)
    if task.getsolsta(mosek.soltype.itr) == mosek.solsta.optimal:
        print("Solution: {xx}".format(xx=list(xx)))
```

The answer is

```
[-0.07838011145615721, 1.1289128998004547, -0.0505327883442975]
```

The dual values $\dot{y}$ of an ACC can be obtained with *Task.getaccdoty* if required.

```
    # Demonstrate retrieving the dual of ACC
    doty = task.getaccdoty(mosek.soltype.itr,
                           0)          # ACC index
    print("Dual of ACC:: {doty}".format(doty=list(doty)))
```

6.2.6 Example ACC2 - more conic constraints

Now that we know how to enter one affine conic constraint (ACC) we will demonstrate a problem with two ACCs. From there it should be clear how to add multiple ACCs. To keep things familiar we will reuse the previous problem, but this time cast it into a conic optimization problem with two ACCs as follows:

$$\begin{array}{ll} \text{maximize} & c^T x \\ \text{subject to} & (\sum_i x_i - 1,\ \gamma,\ Gx + h) \in \{0\} \times \mathcal{Q}^{k+1} \end{array} \tag{6.8}$$

or, using the data from the example:

$$\begin{array}{lll} \text{maximize} & 2x_0 + 3x_1 - x_2 & \\ \text{subject to} & x_0 + x_1 + x_2 - 1 & \in \{0\}, \\ & (0.03, 1.5x_0 + 0.1x_1, 0.3x_0 + 2.1x_2 + 0.1) & \in \mathcal{Q}^3 \end{array}$$

In other words, we transformed the linear constraint into an ACC with the one-point zero domain.

As before, we proceed in three steps. First, we add the variables and create the storage $\mathbf{F}$, $\mathbf{g}$ containing all affine expressions that appear throughout all off the ACCs. It means we will require 4 rows:

$$\mathbf{F} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 1.5 & 0.1 & 0 \\ 0.3 & 0 & 2.1 \end{bmatrix}, \quad \mathbf{g} = \begin{bmatrix} -1 \\ 0.03 \\ 0 \\ 0.1 \end{bmatrix}. \tag{6.9}$$

```
    # Set AFE rows representing the linear constraint
    task.appendafes(1)
    task.putafefrow(0, range(n), [1]*n)
    task.putafeg(0, -1.0)

    # Set AFE rows representing the quadratic constraint
    task.appendafes(k + 1)
    task.putafefrow(2,           # afeidx, row number
                    [0, 1],      # varidx, column numbers
                    [1.5, 0.1]) # values
    task.putafefrow(3,           # afeidx, row number
                    [0, 2],      # varidx, column numbers
                    [0.3, 2.1]) # values

    h     = [0, 0.1]
    gamma = 0.03
    task.putafeg(1, gamma)
    task.putafegslice(2, k+2, h)
```

Next, we add the required domains: the zero domain of dimension 1, and the quadratic cone domain of dimension 3.

```
    # Define domains
    zeroDom = task.appendrzerodomain(1)
    quadDom = task.appendquadraticconedomain(k + 1)
```

Finally, we create both ACCs. The first ACCs picks the 0-th row of $\mathbf{F}, \mathbf{g}$ and places it in the zero domain:

```
    task.appendacc(zeroDom,     # Domain index
                   [0],         # Indices of AFE rows
                   None)        # Ignored
```

The second ACC picks rows $1, 2, 3$ in $\mathbf{F}, \mathbf{g}$ and places them in the quadratic cone domain:

```
    task.appendacc(quadDom,     # Domain index
                   [1,2,3],     # Indices of AFE rows
                   None)        # Ignored
```

The completes the construction and we can solve the problem like before:

Listing 6.3: Full code of example ACC2.

```
import sys, mosek

# Define a stream printer to grab output from MOSEK
def streamprinter(text):
    sys.stdout.write(text)
    sys.stdout.flush()

# Define problem data
n, k = 3, 2
# Only a symbolic constant
inf = 0.0

# Create a task
with mosek.Task() as task:
    # Attach a printer to the task
    task.set_Stream(mosek.streamtype.log, streamprinter)

    # Create n free variables
    task.appendvars(n)
    task.putvarboundsliceconst(0, n, mosek.boundkey.fr, -inf, inf)

    # Set up the objective
    c = [2, 3, -1]
    task.putobjsense(mosek.objsense.maximize)
    task.putclist(range(n), c)

    # Set AFE rows representing the linear constraint
    task.appendafes(1)
    task.putafefrow(0, range(n), [1]*n)
    task.putafeg(0, -1.0)

    # Set AFE rows representing the quadratic constraint
    task.appendafes(k + 1)
    task.putafefrow(2,          # afeidx, row number
                    [0, 1],     # varidx, column numbers
                    [1.5, 0.1]) # values
    task.putafefrow(3,          # afeidx, row number
                    [0, 2],     # varidx, column numbers
                    [0.3, 2.1]) # values

    h     = [0, 0.1]
    gamma = 0.03
    task.putafeg(1, gamma)
    task.putafegslice(2, k+2, h)
```

(continues on next page)

(continued from previous page)

```
    # Define domains
    zeroDom = task.appendrzerodomain(1)
    quadDom = task.appendquadraticconedomain(k + 1)

    # Append affine conic constraints
    task.appendacc(zeroDom,     # Domain index
                   [0],         # Indices of AFE rows
                   None)        # Ignored
    task.appendacc(quadDom,     # Domain index
                   [1,2,3],     # Indices of AFE rows
                   None)        # Ignored

    # Solve and retrieve solution
    task.optimize()
    xx = task.getxx(mosek.soltype.itr)
    if task.getsolsta(mosek.soltype.itr) == mosek.solsta.optimal:
        print("Solution: {xx}".format(xx=list(xx)))
```

We obtain the same result:

```
[-0.07838011145615721, 1.1289128998004547, -0.0505327883442975]
```

6.2.7 Summary and extensions

In this section we presented the most basic usage of the affine expression storage $\mathbf{F}, \mathbf{g}$ to input *affine expressions* used together with *domains* to create *affine conic constraints*. Now we briefly point out additional features of his interface which can be useful in some situations for more demanding users. They will be demonstrated in various examples in other tutorials and case studies in this manual.

- It is important to remember that $\mathbf{F}, \mathbf{g}$ has *only a storage function* and during the ACC construction we can pick an arbitrary list of row indices and place them in a conic domain. It means for example that:
  - It is not necessary to store the AFEs in the same order they will appear in ACCs.
  - The same AFE index can appear more than once in one and/or more conic constraints (this can be used to reduce storage if the same affine expression is used in multiple ACCs).
  - The $\mathbf{F}, \mathbf{g}$ storage can even include rows that are not presently used in any ACC.
- Domains can be reused: multiple ACCs can use the same domain. On the other hand the same type of domain can appear under many `domidx` positions. In this sense the list of created domains also plays only a *storage role*: the domains are only used when they enter an ACC.
- Affine expressions can also contain semidefinite terms, ie. the most general form of an ACC is in fact

  $$Fx + \langle \bar{F}, \overline{X} \rangle + g \in \mathcal{D}$$

  These terms are input into the rows of AFE storage using the functions with infix `afebarf`, creating an additional storage structure $\bar{\mathbf{F}}$.
- The same affine expression storage $\mathbf{F}, \mathbf{g}$ is shared between affine conic and disjunctive constraints (see Sec. 6.9).
- If, on the other hand, the user chooses to always store the AFEs one by one sequentially in the same order as they appear in ACCs then sequential functions such as *Task.appendaccseq* and *Task.appendaccsseq* make it easy to input one or more ACCs by just specifying the starting AFE index and dimension.
- It is possible to add a number of ACCs in one go using *Task.appendaccs*.

- When defining an ACC an additional constant vector b can be provided to modify the constant terms coming from **g** but only for this particular ACC. This could be useful to reduce **F** storage space if, for example, many expressions $f^T x - b_i$ with the same linear part $f^T x$, but varying constant terms b_i, are to be used throughout ACCs.

6.3 Conic Quadratic Optimization

The structure of a typical conic optimization problem is

$$\begin{array}{lccccc} \text{minimize} & & & c^T x + c^f & & \\ \text{subject to} & l^c & \leq & Ax & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x, \\ & & & Fx + g & \in & \mathcal{D}, \end{array}$$

(see Sec. 12 for detailed formulations). We recommend Sec. 6.2 for a tutorial on how problems of that form are represented in MOSEK and what data structures are relevant. Here we discuss how to set-up problems with the **(rotated) quadratic cones**.

MOSEK supports two types of quadratic cones, namely:

- Quadratic cone:

$$\mathcal{Q}^n = \left\{ x \in \mathbb{R}^n : x_0 \geq \sqrt{\sum_{j=1}^{n-1} x_j^2} \right\}.$$

- Rotated quadratic cone:

$$\mathcal{Q}_r^n = \left\{ x \in \mathbb{R}^n : 2x_0 x_1 \geq \sum_{j=2}^{n-1} x_j^2, \quad x_0 \geq 0, \quad x_1 \geq 0 \right\}.$$

For example, consider the following constraint:

$$(x_4, x_0, x_2) \in \mathcal{Q}^3$$

which describes a convex cone in $\mathbb{R}^3$ given by the inequality:

$$x_4 \geq \sqrt{x_0^2 + x_2^2}.$$

For other types of cones supported by **MOSEK**, see Sec. 15.11 and the other tutorials in this chapter. Different cone types can appear together in one optimization problem.

6.3.1 Example CQO1

Consider the following conic quadratic problem which involves some linear constraints, a quadratic cone and a rotated quadratic cone.

$$\begin{array}{lccc} \text{minimize} & x_4 + x_5 + x_6 & & \\ \text{subject to} & x_1 + x_2 + 2x_3 & = & 1, \\ & x_1, x_2, x_3 & \geq & 0, \\ & x_4 \geq \sqrt{x_1^2 + x_2^2}, & & \\ & 2x_5 x_6 \geq x_3^2 & & \end{array} \tag{6.10}$$

The two conic constraints can be expressed in the ACC form as shown in (6.11)

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \in \mathcal{Q}^3 \times \mathcal{Q}_r^3. \tag{6.11}$$

Setting up the linear part

The linear parts (constraints, variables, objective) are set up using exactly the same methods as for linear problems, and we refer to Sec. 6.1 for all the details. The same applies to technical aspects such as defining an optimization task, retrieving the solution and so on.

Setting up the conic constraints

In order to append the conic constraints we first input the matrix $\mathbf{F}$ and vector $\mathbf{g}$ appearing in (6.11). The matrix $\mathbf{F}$ is sparse and we input only its nonzeros using *Task.putafefentrylist*. Since $\mathbf{g}$ is zero, nothing needs to be done about this vector.

Each of the conic constraints is appended using the function *Task.appendacc*. In the first case we append the quadratic cone determined by the first three rows of $\mathbf{F}$ and then the rotated quadratic cone depending on the remaining three rows of $\mathbf{F}$.

```
        # Create a matrix F such that F * x = [x(3),x(0),x(1),x(4),x(5),x(2)]
        task.appendafes(6)
        task.putafefentrylist(range(6),                      # Rows
                              [3, 0, 1, 4, 5, 2],            # Columns
                              [1.0] * 6)

        # Quadratic cone (x(3),x(0),x(1)) \in QUAD_3
        quadcone  = task.appendquadraticconedomain(3)
        task.appendacc(quadcone,                # Domain
                       [0, 1, 2],               # Rows from F
                       None)                    # Unused

        # Rotated quadratic cone (x(4),x(5),x(2)) \in RQUAD_3
        rquadcone = task.appendrquadraticconedomain(3);
        task.appendacc(rquadcone,               # Domain
                       [3, 4, 5],               # Rows from F
                       None)                    # Unused
```

The first argument selects the domain, which must be appended before being used, and must have the dimension matching the number of affine expressions appearing in the constraint. Variants of this method are available to append multiple ACCs at a time. It is also possible to define the matrix $\mathbf{F}$ using a variety of methods (row after row, column by column, individual entries, etc.) similarly as for the linear constraint matrix A.

For a more thorough exposition of the affine expression storage (AFE) matrix $\mathbf{F}$ and vector $\mathbf{g}$ see Sec. 6.2.

Source code

Listing 6.4: Source code solving problem (6.10).

```
import sys
import mosek

# Since the actual value of Infinity is ignores, we define it solely
# for symbolic purposes:
inf = 0.0

# Define a stream printer to grab output from MOSEK
def streamprinter(text):
    sys.stdout.write(text)
    sys.stdout.flush()
```

(continues on next page)

(continued from previous page)

```
def main():
    # Create a task
    with mosek.Task() as task:
        # Attach a printer to the task
        task.set_Stream(mosek.streamtype.log, streamprinter)

        bkc = [mosek.boundkey.fx]
        blc = [1.0]
        buc = [1.0]

        c = [0.0, 0.0, 0.0,
             1.0, 1.0, 1.0]
        bkx = [mosek.boundkey.lo, mosek.boundkey.lo, mosek.boundkey.lo,
               mosek.boundkey.fr, mosek.boundkey.fr, mosek.boundkey.fr]
        blx = [0.0, 0.0, 0.0,
               -inf, -inf, -inf]
        bux = [inf, inf, inf,
               inf, inf, inf]

        asub = [[0], [0], [0]]
        aval = [[1.0], [1.0], [2.0]]

        numvar = len(bkx)
        numcon = len(bkc)
        NUMANZ = 4

        # Append 'numcon' empty constraints.
        # The constraints will initially have no bounds.
        task.appendcons(numcon)

        #Append 'numvar' variables.
        # The variables will initially be fixed at zero (x=0).
        task.appendvars(numvar)

        for j in range(numvar):
            # Set the linear term c_j in the objective.
            task.putcj(j, c[j])
            # Set the bounds on variable j
            # blx[j] <= x_j <= bux[j]
            task.putvarbound(j, bkx[j], blx[j], bux[j])

        for j in range(len(aval)):
            # Input column j of A
            task.putacol(j,                  # Variable (column) index.
                         # Row index of non-zeros in column j.
                         asub[j],
                         aval[j])            # Non-zero Values of column j.

        for i in range(numcon):
            task.putconbound(i, bkc[i], blc[i], buc[i])

        # Input the affine conic constraints
        # Create a matrix F such that F * x = [x(3),x(0),x(1),x(4),x(5),x(2)]
        task.appendafes(6)
        task.putafefentrylist(range(6),                      # Rows
                              [3, 0, 1, 4, 5, 2],            # Columns
```

(continues on next page)

(continued from previous page)

```
                                [1.0] * 6)

        # Quadratic cone (x(3),x(0),x(1)) \in QUAD_3
        quadcone  = task.appendquadraticconedomain(3)
        task.appendacc(quadcone,                # Domain
                       [0, 1, 2],               # Rows from F
                       None)                    # Unused

        # Rotated quadratic cone (x(4),x(5),x(2)) \in RQUAD_3
        rquadcone = task.appendrquadraticconedomain(3);
        task.appendacc(rquadcone,               # Domain
                       [3, 4, 5],               # Rows from F
                       None)                    # Unused

        # Input the objective sense (minimize/maximize)
        task.putobjsense(mosek.objsense.minimize)

        # Optimize the task
        task.optimize()
        task.writedata("cqo1.ptf")
        # Print a summary containing information
        # about the solution for debugging purposes
        task.solutionsummary(mosek.streamtype.msg)
        prosta = task.getprosta(mosek.soltype.itr)
        solsta = task.getsolsta(mosek.soltype.itr)

        # Output a solution
        xx = task.getxx(mosek.soltype.itr)

        if solsta == mosek.solsta.optimal:
            print("Optimal solution: %s" % xx)
        elif solsta == mosek.solsta.dual_infeas_cer:
            print("Primal or dual infeasibility.\n")
        elif solsta == mosek.solsta.prim_infeas_cer:
            print("Primal or dual infeasibility.\n")
        elif mosek.solsta.unknown:
            print("Unknown solution status")
        else:
            print("Other solution status")


# call the main function
try:
    main()
except mosek.MosekException as e:
    print("ERROR: %s" % str(e.errno))
    print("\t%s" % e.msg)
    sys.exit(1)
except:
    import traceback
    traceback.print_exc()
    sys.exit(1)
```

6.4 Power Cone Optimization

The structure of a typical conic optimization problem is

$$\begin{array}{lccccc} \text{minimize} & & & c^T x + c^f & & \\ \text{subject to} & l^c & \leq & Ax & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x, \\ & & & Fx + g & \in & \mathcal{D}, \end{array}$$

(see Sec. 12 for detailed formulations). Here we discuss how to set-up problems with the **primal/dual power cones**.

MOSEK supports the primal and dual power cones, defined as below:

- Primal power cone:

$$\mathcal{P}_n^{\alpha_k} = \left\{ x \in \mathbb{R}^n \ : \ \prod_{i=0}^{n_\ell - 1} x_i^{\beta_i} \geq \sqrt{\sum_{j=n_\ell}^{n-1} x_j^2}, \ x_0 \ldots, x_{n_\ell - 1} \geq 0 \right\}$$

  where $s = \sum_i \alpha_i$ and $\beta_i = \alpha_i / s$, so that $\sum_i \beta_i = 1$.

- Dual power cone:

$$(\mathcal{P}_n^{\alpha_k}) = \left\{ x \in \mathbb{R}^n \ : \ \prod_{i=0}^{n_\ell - 1} \left(\frac{x_i}{\beta_i} \right)^{\beta_i} \geq \sqrt{\sum_{j=n_\ell}^{n-1} x_j^2}, \ x_0 \ldots, x_{n_\ell - 1} \geq 0 \right\}$$

  where $s = \sum_i \alpha_i$ and $\beta_i = \alpha_i / s$, so that $\sum_i \beta_i = 1$.

Perhaps the most important special case is the three-dimensional power cone family:

$$\mathcal{P}_3^{\alpha, 1-\alpha} = \left\{ x \in \mathbb{R}^3 : x_0^\alpha x_1^{1-\alpha} \geq |x_2|, \ x_0, x_1 \geq 0 \right\}.$$

which has the corresponding dual cone:

For example, the conic constraint $(x, y, z) \in \mathcal{P}_3^{0.25, 0.75}$ is equivalent to $x^{0.25} y^{0.75} \geq |z|$, or simply $xy^3 \geq z^4$ with $x, y \geq 0$.

For other types of cones supported by **MOSEK**, see Sec. 15.11 and the other tutorials in this chapter. Different cone types can appear together in one optimization problem.

6.4.1 Example POW1

Consider the following optimization problem which involves powers of variables:

$$\begin{array}{lccc} \text{maximize} & x_0^{0.2} x_1^{0.8} + x_2^{0.4} - x_0 & & \\ \text{subject to} & x_0 + x_1 + \frac{1}{2} x_2 & = & 2, \\ & x_0, x_1, x_2 & \geq & 0. \end{array} \tag{6.12}$$

We convert (6.12) into affine conic form using auxiliary variables as bounds for the power expressions:

$$\begin{array}{lccc} \text{maximize} & x_3 + x_4 - x_0 & & \\ \text{subject to} & x_0 + x_1 + \frac{1}{2} x_2 & = & 2, \\ & (x_0, x_1, x_3) & \in & \mathcal{P}_3^{0.2, 0.8}, \\ & (x_2, 1.0, x_4) & \in & \mathcal{P}_3^{0.4, 0.6}. \end{array} \tag{6.13}$$

The two conic constraints shown in (6.13) can be expressed in the ACC form as shown in (6.14):

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \in \mathcal{P}_3^{0.2, 0.8} \times \mathcal{P}_3^{0.4, 0.6}. \tag{6.14}$$

Setting up the linear part

The linear parts (constraints, variables, objective) are set up using exactly the same methods as for linear problems, and we refer to Sec. 6.1 for all the details. The same applies to technical aspects such as defining an optimization task, retrieving the solution and so on.

Setting up the conic constraints

In order to append the conic constraints we first input the matrix $\mathbf{F}$ and vector $\mathbf{g}$ which together determine all the six affine expressions appearing in the conic constraints of (6.13)

```
        # Append two power cone domains
        pc1 = task.appendprimalpowerconedomain(3, [0.2, 0.8])
        pc2 = task.appendprimalpowerconedomain(3, [4.0, 6.0])

        # Create data structures F,g so that
        #
        #   F * x + g = (x(0), x(1), x(3), x(2), 1.0, x(4))
        #
        task.appendafes(6)
        task.putafefentrylist([0, 1, 2, 3, 5],          # Rows
                              [0, 1, 3, 2, 4],          #Columns
                              [1.0, 1.0, 1.0, 1.0, 1.0])
        task.putafeg(4, 1.0)

        # Append the two conic constraints
        task.appendacc(pc1,                       # Domain
                       [0, 1, 2],                 # Rows from F
                       None)                      # Unused
        task.appendacc(pc2,                       # Domain
                       [3, 4, 5],                 # Rows from F
                       None)                      # Unused
```

Following that, each of the affine conic constraints is appended using the function *Task.appendacc*. The first argument selects the domain, which must be appended before being used, and must have the dimension matching the number of affine expressions appearing in the constraint. In the first case we append the power cone determined by the first three rows of $\mathbf{F}$ and $\mathbf{g}$ while in the second call we use the remaining three rows of $\mathbf{F}$ and $\mathbf{g}$.

Variants of this method are available to append multiple ACCs at a time. It is also possible to define the matrix $\mathbf{F}$ using a variety of methods (row after row, column by column, individual entries, etc.) similarly as for the linear constraint matrix A.

For a more thorough exposition of the affine expression storage (AFE) matrix $\mathbf{F}$ and vector $\mathbf{g}$ see Sec. 6.2.

Source code

Listing 6.5: Source code solving problem (6.12).

```
import sys
import mosek

# Define a stream printer to grab output from MOSEK
def streamprinter(text):
    sys.stdout.write(text)
    sys.stdout.flush()

def main():
    # Only a symbolic constant
```

(continues on next page)

(continued from previous page)

```
    inf = 0.0

    # Create a task
    with mosek.Task() as task:
        # Attach a printer to the task
        task.set_Stream(mosek.streamtype.log, streamprinter)

        csub = [3, 4, 0]
        cval = [1.0, 1.0, -1.0]
        asub = [0, 1, 2]
        aval = [1.0, 1.0, 0.5]
        numvar, numcon = 5, 1          # x,y,z and 2 auxiliary variables for conic␣
↪constraints

        # Append 'numcon' empty constraints.
        # The constraints will initially have no bounds.
        task.appendcons(numcon)

        # Append 'numvar' variables.
        # The variables will initially be fixed at zero (x=0).
        task.appendvars(numvar)

        # Set up the linear part of the problem
        task.putclist(csub, cval)
        task.putarow(0, asub, aval)
        task.putvarboundslice(0, numvar, [mosek.boundkey.fr] * numvar, [inf] * numvar,
↪ [inf] * numvar)
        task.putconbound(0, mosek.boundkey.fx, 2.0, 2.0)

        # Input the conic constraints
        # Append two power cone domains
        pc1 = task.appendprimalpowerconedomain(3, [0.2, 0.8])
        pc2 = task.appendprimalpowerconedomain(3, [4.0, 6.0])

        # Create data structures F,g so that
        #
        #   F * x + g = (x(0), x(1), x(3), x(2), 1.0, x(4))
        #
        task.appendafes(6)
        task.putafefentrylist([0, 1, 2, 3, 5],          # Rows
                              [0, 1, 3, 2, 4],          #Columns
                              [1.0, 1.0, 1.0, 1.0, 1.0])
        task.putafeg(4, 1.0)

        # Append the two conic constraints
        task.appendacc(pc1,                       # Domain
                       [0, 1, 2],                 # Rows from F
                       None)                      # Unused
        task.appendacc(pc2,                       # Domain
                       [3, 4, 5],                 # Rows from F
                       None)                      # Unused

        # Input the objective sense (minimize/maximize)
        task.putobjsense(mosek.objsense.maximize)

        # Optimize the task
```

(continues on next page)

(continued from previous page)

```
        task.optimize()

        # Print a summary containing information
        # about the solution for debugging purposes
        task.solutionsummary(mosek.streamtype.msg)
        prosta = task.getprosta(mosek.soltype.itr)
        solsta = task.getsolsta(mosek.soltype.itr)

        # Output a solution
        xx = task.getxx(mosek.soltype.itr)

        if solsta == mosek.solsta.optimal:
            print("Optimal solution: %s" % xx[0:3])
        elif solsta == mosek.solsta.dual_infeas_cer:
            print("Primal or dual infeasibility.\n")
        elif solsta == mosek.solsta.prim_infeas_cer:
            print("Primal or dual infeasibility.\n")
        elif mosek.solsta.unknown:
            print("Unknown solution status")
        else:
            print("Other solution status")


# call the main function
try:
    main()
except mosek.MosekException as e:
    print("ERROR: %s" % str(e.code))
    if msg is not None:
        print("\t%s" % e.msg)
        sys.exit(1)
except:
    import traceback
    traceback.print_exc()
    sys.exit(1)
```

6.5 Conic Exponential Optimization

The structure of a typical conic optimization problem is

$$\begin{array}{lccccc} \text{minimize} & & & c^T x + c^f & & \\ \text{subject to} & l^c & \leq & Ax & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x, \\ & & & Fx + g & \in & \mathcal{D}, \end{array}$$

(see Sec. 12 for detailed formulations). We recommend Sec. 6.2 for a tutorial on how problems of that form are represented in MOSEK and what data structures are relevant. Here we discuss how to set-up problems with the **primal/dual exponential cones**.

MOSEK supports two exponential cones, namely:

- Primal exponential cone:

$$K_{\exp} = \left\{x \in \mathbb{R}^3 : x_0 \geq x_1 \exp(x_2/x_1),\ x_0, x_1 \geq 0\right\}.$$

- Dual exponential cone:

$$K^*_{\exp} = \left\{ s \in \mathbb{R}^3 : s_0 \geq -s_2 e^{-1} \exp(s_1/s_2),\ s_2 \leq 0, s_0 \geq 0 \right\}.$$

For example, consider the following constraint:

$$(x_4, x_0, x_2) \in K_{\exp}$$

which describes a convex cone in $\mathbb{R}^3$ given by the inequalities:

$$x_4 \geq x_0 \exp(x_2/x_0),\ x_0, x_4 \geq 0.$$

For other types of cones supported by **MOSEK**, see Sec. 15.11 and the other tutorials in this chapter. Different cone types can appear together in one optimization problem.

6.5.1 Example CEO1

Consider the following basic conic exponential problem which involves some linear constraints and an exponential inequality:

$$\begin{array}{lrcl} \text{minimize} & x_0 + x_1 & & \\ \text{subject to} & x_0 + x_1 + x_2 & = & 1, \\ & x_0 & \geq & x_1 \exp(x_2/x_1), \\ & x_0, x_1 & \geq & 0. \end{array} \tag{6.15}$$

The affine conic form of (6.15) is:

$$\begin{array}{lrcl} \text{minimize} & x_0 + x_1 & & \\ \text{subject to} & x_0 + x_1 + x_2 & = & 1, \\ & Ix & \in & K_{\exp}, \\ & x & \in & \mathbb{R}^3. \end{array} \tag{6.16}$$

where I is the 3×3 identity matrix.

Setting up the linear part

The linear parts (constraints, variables, objective) are set up using exactly the same methods as for linear problems, and we refer to Sec. 6.1 for all the details. The same applies to technical aspects such as defining an optimization task, retrieving the solution and so on.

Setting up the conic constraints

In order to append the conic constraints we first input the sparse identity matrix **F** as indicated by (6.16).

The affine conic constraint is then appended using the function *Task.appendacc*, with the primal exponential domain and the list of **F** rows, in this case consisting of all rows in their natural order.

```
# Create a 3x3 identity matrix F
task.appendafes(3)
task.putafefentrylist([0, 1, 2],          # Rows
                      [0, 1, 2],          # Columns
                      [1.0] * 3)

# Exponential cone (x(0),x(1),x(2)) \in EXP
expdomain  = task.appendprimalexpconedomain()
task.appendacc(expdomain,                 # Domain
               [0, 1, 2],                 # Rows from F
               None)                      # Unused
```

The first argument selects the domain, which must be appended before being used, and must have the dimension matching the number of affine expressions appearing in the constraint. Variants of this method are available to append multiple ACCs at a time. It is also possible to define the matrix $\mathbf{F}$ using a variety of methods (row after row, column by column, individual entries, etc.) similarly as for the linear constraint matrix A.

For a more thorough exposition of the affine expression storage (AFE) matrix $\mathbf{F}$ and vector $\mathbf{g}$ see Sec. 6.2.

Source code

Listing 6.6: Source code solving problem (6.15).

```
import sys
import mosek

# Define a stream printer to grab output from MOSEK
def streamprinter(text):
    sys.stdout.write(text)
    sys.stdout.flush()

def main():

    # Only a symbolic constant
    inf = 0.0

    # Create a task
    with mosek.Task() as task:
        # Attach a printer to the task
        task.set_Stream(mosek.streamtype.log, streamprinter)

        c = [1.0, 1.0, 0.0]
        a = [1.0, 1.0, 1.0]
        numvar, numcon = 3, 1

        # Append 'numcon' empty constraints.
        # The constraints will initially have no bounds.
        task.appendcons(numcon)

        # Append 'numvar' variables.
        # The variables will initially be fixed at zero (x=0).
        task.appendvars(numvar)

        # Set up the linear part of the problem
        task.putcslice(0, numvar, c)
        task.putarow(0, [0, 1, 2], a)
        task.putvarboundslice(0, numvar, [mosek.boundkey.fr] * numvar, [inf] * numvar,
↪ [inf] * numvar)
        task.putconbound(0, mosek.boundkey.fx, 1.0, 1.0)

        # Add a conic constraint
        # Create a 3x3 identity matrix F
        task.appendafes(3)
        task.putafefentrylist([0, 1, 2],          # Rows
                              [0, 1, 2],          # Columns
                              [1.0] * 3)

        # Exponential cone (x(0),x(1),x(2)) \in EXP
```

(continues on next page)

(continued from previous page)

```
        expdomain  = task.appendprimalexpconedomain()
        task.appendacc(expdomain,                # Domain
                       [0, 1, 2],                # Rows from F
                       None)                     # Unused

        # Input the objective sense (minimize/maximize)
        task.putobjsense(mosek.objsense.minimize)
        task.writedata("ceo1-py.ptf")

        # Optimize the task
        task.optimize()
        # Print a summary containing information
        # about the solution for debugging purposes
        task.solutionsummary(mosek.streamtype.msg)
        prosta = task.getprosta(mosek.soltype.itr)
        solsta = task.getsolsta(mosek.soltype.itr)

        # Output a solution
        xx = task.getxx(mosek.soltype.itr)

        if solsta == mosek.solsta.optimal:
            print("Optimal solution: %s" % xx)
        elif solsta == mosek.solsta.dual_infeas_cer:
            print("Primal or dual infeasibility.\n")
        elif solsta == mosek.solsta.prim_infeas_cer:
            print("Primal or dual infeasibility.\n")
        elif mosek.solsta.unknown:
            print("Unknown solution status")
        else:
            print("Other solution status")


# call the main function
try:
    main()
except mosek.MosekException as e:
    print("ERROR: %s" % str(e.code))
    if msg is not None:
        print("\t%s" % e.msg)
        sys.exit(1)
except:
    import traceback
    traceback.print_exc()
    sys.exit(1)
```

6.6 Geometric Programming

Geometric programs (GP) are a particular class of optimization problems which can be expressed in special polynomial form as positive sums of generalized monomials. More precisely, a geometric problem in canonical form is

$$\begin{array}{lll} \text{minimize} & f_0(x) & \\ \text{subject to} & f_i(x) \leq 1, & i = 1, \ldots, m, \\ & x_j > 0, & j = 1, \ldots, n, \end{array} \tag{6.17}$$

where each $f_0, \ldots, f_m$ is a *posynomial*, that is a function of the form

$$f(x) = \sum_k c_k x_1^{\alpha_{k1}} x_2^{\alpha_{k2}} \cdots x_n^{\alpha_{kn}}$$

with arbitrary real α_{ki} and $c_k > 0$. The standard way to formulate GPs in convex form is to introduce a variable substitution

$$x_i = \exp(y_i).$$

Under this substitution all constraints in a GP can be reduced to the form

$$\log(\sum_k \exp(a_k^T y + b_k)) \leq 0 \tag{6.18}$$

involving a *log-sum-exp* bound. Moreover, constraints involving only a single monomial in x can be even more simply written as a linear inequality:

$$a_k^T y + b_k \leq 0$$

We refer to the **MOSEK** Modeling Cookbook and to [BKVH07] for more details on this reformulation. A geometric problem formulated in convex form can be entered into **MOSEK** with the help of exponential cones.

6.6.1 Example GP1

The following problem comes from [BKVH07]. Consider maximizing the volume of a $h \times w \times d$ box subject to upper bounds on the area of the floor and of the walls and bounds on the ratios h/w and d/w:

$$\begin{array}{lrl} \text{maximize} & hwd & \\ \text{subject to} & 2(hw + hd) & \leq A_{\text{wall}}, \\ & wd & \leq A_{\text{floor}}, \\ & \alpha & \leq h/w \leq \beta, \\ & \gamma & \leq d/w \leq \delta. \end{array} \tag{6.19}$$

The decision variables in the problem are h, w, d. We make a substitution

$$h = \exp(x), w = \exp(y), d = \exp(z)$$

after which (6.19) becomes

$$\begin{array}{ll} \text{maximize} & x + y + z \\ \text{subject to} & \log(\exp(x + y + \log(2/A_{\text{wall}})) + \exp(x + z + \log(2/A_{\text{wall}}))) \leq 0, \\ & y + z \leq \log(A_{\text{floor}}), \\ & \log(\alpha) \leq x - y \leq \log(\beta), \\ & \log(\gamma) \leq z - y \leq \log(\delta). \end{array} \tag{6.20}$$

Next, we demonstrate how to implement a log-sum-exp constraint (6.18). It can be written as:

$$\begin{array}{l} u_k \geq \exp(a_k^T y + b_k), \quad (\text{equiv. } (u_k, 1, a_k^T y + b_k) \in K_{\exp}), \\ \sum_k u_k = 1. \end{array} \tag{6.21}$$

This presentation requires one extra variable u_k for each monomial appearing in the original posynomial constraint. In this case the affine conic constraints (ACC, see Sec. 6.2) take the form:

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ u_1 \\ u_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ \log(2/A_{\text{wall}}) \\ 0 \\ 1 \\ \log(2/A_{\text{wall}}) \end{bmatrix} \in K_{\text{exp}} \times K_{\text{exp}}.$$

As a matter of demonstration we will also add the constraint

$$u_1 + u_2 - 1 = 0$$

as an affine conic constraint. It means that to define the all the ACCs we need to produce the following affine expressions (AFE) and store them:

$$u_1,\ u_2,\ x + y + \log(2/A_{\text{wall}}),\ x + z + \log(2/A_{\text{wall}}),\ 1.0,\ u_1 + u_2 - 1.0.$$

We implement it by adding all the affine expressions (AFE) and then picking the ones required for each ACC:

Listing 6.7: Implementation of log-sum-exp as in (6.21).

```
        # Affine expressions appearing in affine conic constraints
        # in this order:
        # u1, u2, x+y+log(2/Awall), x+z+log(2/Awall), 1.0, u1+u2-1.0
        numafe    = 6
        u1, u2 = 3, 4     # Indices of slack variables
        afeidx = [0, 1, 2, 2, 3, 3, 5, 5]
        varidx = [u1, u2, x, y, x, z, u1, u2]
        fval   = [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
        gfull  = [0, 0, log(2/Aw), log(2/Aw), 1.0, -1.0]

        # New variables u1, u2
        task.appendvars(2)
        task.putvarboundsliceconst(u1, u2+1, boundkey.fr, -inf, inf)

        # Append affine expressions
        task.appendafes(numafe)
        task.putafefentrylist(afeidx, varidx, fval)
        task.putafegslice(0, numafe, gfull)

        # Two affine conic constraints
        expdom = task.appendprimalexpconedomain()

        # (u1, 1, x+y+log(2/Awall)) \in EXP
        task.appendacc(expdom, [0, 4, 2], None)

        # (u2, 1, x+z+log(2/Awall)) \in EXP
        task.appendacc(expdom, [1, 4, 3], None)

        # The constraint u1+u2-1 \in \ZERO is added also as an ACC
        task.appendacc(task.appendrzerodomain(1), [5], None)
```

We can now use this function to assemble all constraints in the model. The linear part of the problem is entered as in Sec. 6.1.

Listing 6.8: Source code solving problem (6.20).

```
def max_volume_box(Aw, Af, alpha, beta, gamma, delta):
    # Basic dimensions of our problem
    numvar    = 3       # Variables in original problem
    x, y, z   = 0, 1, 2 # Indices of variables
    numcon    = 3       # Linear constraints in original problem

    # Linear part of the problem
    cval  = [1, 1, 1]
    asubi = [0, 0, 1, 1, 2, 2]
    asubj = [y, z, x, y, z, y]
    aval  = [1.0, 1.0, 1.0, -1.0, 1.0, -1.0]
    bkc   = [boundkey.up, boundkey.ra, boundkey.ra]
    blc   = [-inf, log(alpha), log(gamma)]
    buc   = [log(Af), log(beta), log(delta)]

    with Task() as task:
        task.set_Stream(streamtype.log, streamprinter)

        # Add variables and constraints
        task.appendvars(numvar)
        task.appendcons(numcon)

        # Objective is the sum of three first variables
        task.putobjsense(objsense.maximize)
        task.putcslice(0, numvar, cval)
        task.putvarboundsliceconst(0, numvar, boundkey.fr, -inf, inf)

        # Add the three linear constraints
        task.putaijlist(asubi, asubj, aval)
        task.putconboundslice(0, numvar, bkc, blc, buc)

        # Affine expressions appearing in affine conic constraints
        # in this order:
        # u1, u2, x+y+log(2/Awall), x+z+log(2/Awall), 1.0, u1+u2-1.0
        numafe    = 6
        u1, u2 = 3, 4     # Indices of slack variables
        afeidx = [0, 1, 2, 2, 3, 3, 5, 5]
        varidx = [u1, u2, x, y, x, z, u1, u2]
        fval   = [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
        gfull  = [0, 0, log(2/Aw), log(2/Aw), 1.0, -1.0]

        # New variables u1, u2
        task.appendvars(2)
        task.putvarboundsliceconst(u1, u2+1, boundkey.fr, -inf, inf)

        # Append affine expressions
        task.appendafes(numafe)
        task.putafefentrylist(afeidx, varidx, fval)
        task.putafegslice(0, numafe, gfull)

        # Two affine conic constraints
        expdom = task.appendprimalexpconedomain()

        # (u1, 1, x+y+log(2/Awall)) \in EXP
        task.appendacc(expdom, [0, 4, 2], None)
```

(continues on next page)

(continued from previous page)

```
        # (u2, 1, x+z+log(2/Awall)) \in EXP
        task.appendacc(expdom, [1, 4, 3], None)

        # The constraint u1+u2-1 \in \ZERO is added also as an ACC
        task.appendacc(task.appendrzerodomain(1), [5], None)

        # Solve and map to original h, w, d
        task.optimize()
        task.writedata("gp1.ptf");
        xyz = task.getxxslice(soltype.itr, 0, numvar)
        return exp(xyz)
```

Given sample data we obtain the solution h, w, d as follows:

Listing 6.9: Sample data for problem (6.19).

```
Aw, Af, alpha, beta, gamma, delta = 200.0, 50.0, 2.0, 10.0, 2.0, 10.0
h,w,d = max_volume_box(Aw, Af, alpha, beta, gamma, delta)
print("h={0:.3f}, w={1:.3f}, d={2:.3f}".format(h, w, d))
```

6.7 Semidefinite Optimization

Semidefinite optimization is a generalization of conic optimization, allowing the use of matrix variables belonging to the convex cone of positive semidefinite matrices

$$\mathcal{S}^r_+ = \left\{ X \in \mathcal{S}^r : z^T X z \geq 0, \quad \forall z \in \mathbb{R}^r \right\},$$

where $\mathcal{S}^r$ is the set of $r \times r$ real-valued symmetric matrices.

MOSEK can solve semidefinite optimization problems stated in the **primal** form,

$$\begin{array}{lrcc} \text{minimize} & & \sum_{j=0}^{p-1} \langle \overline{C}_j, \overline{X}_j \rangle + \sum_{j=0}^{n-1} c_j x_j + c^f & \\ \text{subject to} & l^c_i \leq & \sum_{j=0}^{p-1} \langle \overline{A}_{ij}, \overline{X}_j \rangle + \sum_{j=0}^{n-1} a_{ij} x_j & \leq u^c_i, \quad i = 0, \ldots, m-1, \\ & & \sum_{j=0}^{p-1} \langle \overline{F}_{ij}, \overline{X}_j \rangle + \sum_{j=0}^{n-1} f_{ij} x_j + g_i & \in \mathcal{K}_i, \quad i = 0, \ldots, q-1, \\ & l^x_j \leq & x_j & \leq u^x_j, \quad j = 0, \ldots, n-1, \\ & & x \in \mathcal{K}, \overline{X}_j \in \mathcal{S}^{r_j}_+, & \quad j = 0, \ldots, p-1 \end{array} \tag{6.22}$$

where the problem has p symmetric positive semidefinite variables $\overline{X}_j \in \mathcal{S}^{r_j}_+$ of dimension r_j. The symmetric coefficient matrices $\overline{C}_j \in \mathcal{S}^{r_j}$ and $\overline{A}_{i,j} \in \mathcal{S}^{r_j}$ are used to specify PSD terms in the linear objective and the linear constraints, respectively. The symmetric coefficient matrices $\overline{F}_{i,j} \in \mathcal{S}^{r_j}$ are used to specify PSD terms in the affine conic constraints. Note that q ((6.22)) is the total dimension of all the cones, i.e. $q = \dim(\mathcal{K}_1 \times \ldots \times \mathcal{K}_k)$, given there are k ACCs. We use standard notation for the matrix inner product, i.e., for $A, B \in \mathbb{R}^{m \times n}$ we have

$$\langle A, B \rangle := \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} A_{ij} B_{ij}.$$

In addition to the primal form presented above, semidefinite problems can be expressed in their **dual** form. Constraints in this form are usually called **linear matrix inequalities** (LMIs). LMIs can be easily specified in **MOSEK** using the vectorized positive semidefinite cone which is defined as:

- Vectorized semidefinite domain:

$$\mathcal{S}^{d,\text{vec}}_+ = \left\{ (x_1, \ldots, x_{d(d+1)/2}) \in \mathbb{R}^n \ : \ \text{sMat}(x) \in \mathcal{S}^d_+ \right\},$$

where $n = d(d+1)/2$ and,

$$\mathrm{sMat}(x) = \begin{bmatrix} x_1 & x_2/\sqrt{2} & \cdots & x_d/\sqrt{2} \\ x_2/\sqrt{2} & x_{d+1} & \cdots & x_{2d-1}/\sqrt{2} \\ \cdots & \cdots & \cdots & \cdots \\ x_d/\sqrt{2} & x_{2d-1}/\sqrt{2} & \cdots & x_{d(d+1)/2} \end{bmatrix},$$

or equivalently

$$\mathcal{S}_+^{d,\mathrm{vec}} = \left\{ \mathrm{sVec}(X) \ : \ X \in \mathcal{S}_+^d \right\},$$

where

$$\mathrm{sVec}(X) = (X_{11}, \sqrt{2}X_{21}, \ldots, \sqrt{2}X_{d1}, X_{22}, \sqrt{2}X_{32}, \ldots, X_{dd}).$$

In other words, the domain consists of vectorizations of the lower-triangular part of a positive semidefinite matrix, with the non-diagonal elements additionally rescaled. LMIs can be expressed by restricting appropriate affine expressions to this cone type.

For other types of cones supported by **MOSEK**, see Sec. 15.11 and the other tutorials in this chapter. Different cone types can appear together in one optimization problem.

We demonstrate the setup of semidefinite variables and their coefficient matrices in the following examples:

- Sec. 6.7.1: A problem with one semidefinite variable and linear and conic constraints.
- Sec. 6.7.2: A problem with two semidefinite variables with a linear constraint and bound.
- Sec. 6.7.3: A problem with linear matrix inequalities and the vectorized semidefinite domain.

6.7.1 Example SDO1

We consider the simple optimization problem with semidefinite and conic quadratic constraints:

$$\begin{array}{lll} \text{minimize} & \left\langle \begin{bmatrix} 2 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 2 \end{bmatrix}, \overline{X} \right\rangle + x_0 & \\ \text{subject to} & \left\langle \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \overline{X} \right\rangle + x_0 & = \quad 1, \\ & \left\langle \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \overline{X} \right\rangle + x_1 + x_2 & = \quad 1/2, \\ & x_0 \geq \sqrt{{x_1}^2 + {x_2}^2}, & \overline{X} \succeq 0, \end{array} \tag{6.23}$$

The problem description contains a 3-dimensional symmetric semidefinite variable which can be written explicitly as:

$$\overline{X} = \begin{bmatrix} \overline{X}_{00} & \overline{X}_{10} & \overline{X}_{20} \\ \overline{X}_{10} & \overline{X}_{11} & \overline{X}_{21} \\ \overline{X}_{20} & \overline{X}_{21} & \overline{X}_{22} \end{bmatrix} \in \mathcal{S}_+^3,$$

and an affine conic constraint (ACC) $(x_0, x_1, x_2) \in \mathcal{Q}^3$. The objective is to minimize

$$2(\overline{X}_{00} + \overline{X}_{10} + \overline{X}_{11} + \overline{X}_{21} + \overline{X}_{22}) + x_0,$$

subject to the two linear constraints

$$\begin{array}{rcl} \overline{X}_{00} + \overline{X}_{11} + \overline{X}_{22} + x_0 & = & 1, \\ \overline{X}_{00} + \overline{X}_{11} + \overline{X}_{22} + 2(\overline{X}_{10} + \overline{X}_{20} + \overline{X}_{21}) + x_1 + x_2 & = & 1/2. \end{array}$$

Setting up the linear and conic part

The linear and conic parts (constraints, variables, objective, ACC) are set up using the methods described in the relevant tutorials; Sec. 6.1, Sec. 6.2. Here we only discuss the aspects directly involving semidefinite variables.

Appending semidefinite variables

First, we need to declare the number of semidefinite variables in the problem, similarly to the number of linear variables and constraints. This is done with the function *Task.appendbarvars*.

```
task.appendbarvars(BARVARDIM)
```

Appending coefficient matrices

Coefficient matrices $\overline{C}_j$ and $\overline{A}_{ij}$ are constructed as weighted combinations of sparse symmetric matrices previously appended with the function *Task.appendsparsesymmat*.

```
symc = task.appendsparsesymmat(BARVARDIM[0],
                               barci,
                               barcj,
                               barcval)

syma0 = task.appendsparsesymmat(BARVARDIM[0],
                                barai[0],
                                baraj[0],
                                baraval[0])

syma1 = task.appendsparsesymmat(BARVARDIM[0],
                                barai[1],
                                baraj[1],
                                baraval[1])
```

The arguments specify the dimension of the symmetric matrix, followed by its description in the sparse triplet format. Only lower-triangular entries should be included. The function produces a unique index of the matrix just entered in the collection of all coefficient matrices defined by the user.

After one or more symmetric matrices have been created using *Task.appendsparsesymmat*, we can combine them to set up the objective matrix coefficient $\overline{C}_j$ using *Task.putbarcj*, which forms a linear combination of one or more symmetric matrices. In this example we form the objective matrix directly, i.e. as a weighted combination of a single symmetric matrix.

```
task.putbarcj(0, [symc], [1.0])
```

Similarly, a constraint matrix coefficient $\overline{A}_{ij}$ is set up by the function *Task.putbaraij*.

```
task.putbaraij(0, 0, [syma0], [1.0])
task.putbaraij(1, 0, [syma1], [1.0])
```

Retrieving the solution

After the problem is solved, we read the solution using *Task.getbarxj*:

```
            barx = task.getbarxj(mosek.soltype.itr, 0)
```

The function returns the half-vectorization of $\overline{X}_j$ (the lower triangular part stacked as a column vector), where the semidefinite variable index j is passed as an argument.

Source code

Listing 6.10: Source code solving problem (6.23).

```
import sys
import mosek

# Since the value of infinity is ignored, we define it solely
# for symbolic purposes
inf = 0.0

# Define a stream printer to grab output from MOSEK
def streamprinter(text):
    sys.stdout.write(text)
    sys.stdout.flush()


def main():
    # Create a task object and attach log stream printer
    with mosek.Task() as task:
        task.set_Stream(mosek.streamtype.log, streamprinter)

        # Bound keys for constraints
        bkc = [mosek.boundkey.fx,
               mosek.boundkey.fx]

        # Bound values for constraints
        blc = [1.0, 0.5]
        buc = [1.0, 0.5]

        # Below is the sparse representation of the A
        # matrix stored by row.
        asub = [[0],
                [1, 2]]
        aval = [[1.0],
                [1.0, 1.0]]

        barci = [0, 1, 1, 2, 2]
        barcj = [0, 0, 1, 1, 2]
        barcval = [2.0, 1.0, 2.0, 1.0, 2.0]

        barai = [[0, 1, 2],
                 [0, 1, 2, 1, 2, 2]]
        baraj = [[0, 1, 2],
                 [0, 0, 0, 1, 1, 2]]
        baraval = [[1.0, 1.0, 1.0],
                   [1.0, 1.0, 1.0, 1.0, 1.0, 1.0]]

        numvar = 3
```

(continues on next page)

(continued from previous page)

```
        numcon = len(bkc)
        BARVARDIM = [3]

        # Append 'numvar' variables.
        # The variables will initially be fixed at zero (x=0).
        task.appendvars(numvar)

        # Append 'numcon' empty constraints.
        # The constraints will initially have no bounds.
        task.appendcons(numcon)

        # Append matrix variables of sizes in 'BARVARDIM'.
        # The variables will initially be fixed at zero.
        task.appendbarvars(BARVARDIM)

        # Set the linear term c_0 in the objective.
        task.putcj(0, 1.0)

        for j in range(numvar):
            # Set the bounds on variable j
            # blx[j] <= x_j <= bux[j]
            task.putvarbound(j, mosek.boundkey.fr, -inf, +inf)

        for i in range(numcon):
            # Set the bounds on constraints.
            # blc[i] <= constraint_i <= buc[i]
            task.putconbound(i, bkc[i], blc[i], buc[i])

            # Input row i of A
            task.putarow(i,                   # Constraint (row) index.
                         asub[i],             # Column index of non-zeros in␣
↪constraint i.
                         aval[i])             # Non-zero values of row i.

        # Add the quadratic cone constraint
        task.appendafes(3)
        # Diagonal F matrix
        task.putafefentrylist(range(3), range(3), [1.0]*3)
        task.appendaccseq(task.appendquadraticconedomain(3), 0, None)

        symc = task.appendsparsesymmat(BARVARDIM[0],
                                       barci,
                                       barcj,
                                       barcval)

        syma0 = task.appendsparsesymmat(BARVARDIM[0],
                                        barai[0],
                                        baraj[0],
                                        baraval[0])

        syma1 = task.appendsparsesymmat(BARVARDIM[0],
                                        barai[1],
                                        baraj[1],
                                        baraval[1])

        task.putbarcj(0, [symc], [1.0])
```

(continues on next page)

(continued from previous page)

```
        task.putbaraij(0, 0, [syma0], [1.0])
        task.putbaraij(1, 0, [syma1], [1.0])

        # Input the objective sense (minimize/maximize)
        task.putobjsense(mosek.objsense.minimize)

        # Solve the problem and print summary
        task.optimize()
        task.solutionsummary(mosek.streamtype.msg)

        # Get status information about the solution
        prosta = task.getprosta(mosek.soltype.itr)
        solsta = task.getsolsta(mosek.soltype.itr)

        if (solsta == mosek.solsta.optimal):
            xx = task.getxx(mosek.soltype.itr)
            barx = task.getbarxj(mosek.soltype.itr, 0)

            print("Optimal solution:\nx=%s\nbarx=%s" % (xx, barx))
        elif (solsta == mosek.solsta.dual_infeas_cer or
              solsta == mosek.solsta.prim_infeas_cer):
            print("Primal or dual infeasibility certificate found.\n")
        elif solsta == mosek.solsta.unknown:
            print("Unknown solution status")
        else:
            print("Other solution status")


# call the main function
try:
    main()
except mosek.MosekException as e:
    print("ERROR: %s" % str(e.errno))
    if e.msg is not None:
        print("\t%s" % e.msg)
        sys.exit(1)
except:
    import traceback
    traceback.print_exc()
    sys.exit(1)
```

6.7.2 Example SDO2

We now demonstrate how to define more than one semidefinite variable using the following problem with two matrix variables and two types of constraints:

$$\begin{array}{lll} \text{minimize} & \langle C_1, \overline{X}_1 \rangle + \langle C_2, \overline{X}_2 \rangle & \\ \text{subject to} & \langle A_1, \overline{X}_1 \rangle + \langle A_2, \overline{X}_2 \rangle & = \quad b, \\ & (\overline{X}_2)_{01} & \leq \quad k, \\ & \overline{X}_1, \overline{X}_2 & \succeq \quad 0. \end{array} \tag{6.24}$$

In our example $\dim(\overline{X}_1) = 3$, $\dim(\overline{X}_2) = 4$, $b = 23$, $k = -3$ and

$$C_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 6 \end{bmatrix}, A_1 = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 2 \end{bmatrix},$$

$$C_2 = \begin{bmatrix} 1 & -3 & 0 & 0 \\ -3 & 2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, A_2 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -3 \end{bmatrix},$$

are constant symmetric matrices.

Note that this problem does not contain any scalar variables, but they could be added in the same fashion as in Sec. 6.7.1.

Other than in Sec. 6.7.1 we don't append coefficient matrices separately but we directly input all nonzeros in each constraint and all nonzeros in the objective at once. Every term of the form $(\overline{A}_{i,j})_{k,l}(\overline{X}_j)_{k,l}$ is determined by four indices (i, j, k, l) and a coefficient value $v = (\overline{A}_{i,j})_{k,l}$. Here i is the number of the constraint in which the term appears, j is the index of the semidefinite variable it involves and (k, l) is the position in that variable. This data is passed in the call to *Task.putbarablocktriplet*. Note that only the lower triangular part should be specified explicitly, that is one always has $k \geq l$. Semidefinite terms $(\overline{C}_j)_{k,l}(\overline{X}_j)_{k,l}$ of the objective are specified in the same way in *Task.putbarcblocktriplet* but only include (j, k, l) and v.

For explanations of other data structures used in the example see Sec. 6.7.1.

The code representing the above problem is shown below.

Listing 6.11: Implementation of model (6.24).

```
# Create a task object and attach log stream printer
with Task() as task:
    # Set log handler for debugging ootput
    task.set_Stream(streamtype.log, streamprinter)

    # Append two symmetric variables of dimension 3, 4
    barvardims = [3, 4]
    task.appendbarvars(barvardims)

    # Semidefinite part of objective function
    task.putbarcblocktriplet(
        [0]*len(C1_v) + [1]*len(C2_v), # Which SDP variable (j)
        C1_k + C2_k,                   # Entries: (k,l)->v
        C1_l + C2_l,
        C1_v + C2_v,
        )

    # Append two constraints
    task.appendcons(2)

    # First constraint (equality)
    task.putbarablocktriplet(
        [0]*(len(A1_v)+len(A2_v)),     # Which constraint (i = 0)
        [0]*len(A1_v) + [1]*len(A2_v), # Which SDP variable (j)
        A1_k + A2_k,                   # Entries: (k,l)->v
        A1_l + A2_l,
        A1_v + A2_v,
        )

    # Second constraint (X2)_{1,2} <= k
    task.putbarablocktriplet(
        [1],                           # Which constraint (i = 1)
        [1],                           # Which SDP variable (j = 1)
        [1], [0], [0.5]                # Entries: (k,l)->v
        )

    # Set bounds for constraints
    task.putconboundlist([0,1], [boundkey.fx, boundkey.up],
```

(continues on next page)

(continued from previous page)

```
                                [b, -inf],
                                [b, k])

    # Write the problem for human inspection
    task.writedata("test.ptf")

    # Optimize
    task.optimize()
    task.solutionsummary(streamtype.msg)

    # Get status information about the solution
    solsta = task.getsolsta(soltype.itr)

    if solsta == solsta.optimal:
        # Assuming the optimization succeeded read solution
        print("Solution (lower-triangular part vectorized): ")
        for i in range(2):
            X = task.getbarxj(soltype.itr, i)
            print("X{i} = {X}".format(i=i, X=X))

    elif (solsta == solsta.dual_infeas_cer or
          solsta == solsta.prim_infeas_cer):
        print("Primal or dual infeasibility certificate found.\n")
    elif solsta == solsta.unknown:
        print("Unknown solution status")
    else:
        print("Other solution status")
```

6.7.3 Example SDO_LMI: Linear matrix inequalities and the vectorized semidefinite domain

The standard form of a semidefinite problem is usually either based on semidefinite variables (primal form) or on linear matrix inequalities (dual form). However, **MOSEK** allows mixing of these two forms, as shown in (6.25)

$$\begin{array}{lll} \text{minimize} & \left\langle \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \overline{X} \right\rangle + x_0 + x_1 + 1 & \\ \text{subject to} & \left\langle \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \overline{X} \right\rangle - x_0 - x_1 & \in \mathbb{R}^1_{\geq 0}, \\ & x_0 \begin{bmatrix} 0 & 1 \\ 1 & 3 \end{bmatrix} + x_1 \begin{bmatrix} 3 & 1 \\ 1 & 0 \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \succeq 0, \\ & \overline{X} \succeq 0. & \end{array} \tag{6.25}$$

The first affine expression is restricted to a linear domain and could also be modelled as a linear constraint (instead of an ACC). The lower triangular part of the linear matrix inequality (second constraint) can be vectorized and restricted to the *domaintype.svec_psd_cone*. This allows us to express the constraints in (6.25) as the affine conic constraints shown in (6.26).

$$\begin{array}{lll} \left\langle \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \overline{X} \right\rangle + \begin{bmatrix} -1 & -1 \end{bmatrix} x + \begin{bmatrix} 0 \end{bmatrix} & \in \mathbb{R}^1_{\geq 0}, \\ \begin{bmatrix} 0 & 3 \\ \sqrt{2} & \sqrt{2} \\ 3 & 0 \end{bmatrix} x + \begin{bmatrix} -1 \\ 0 \\ -1 \end{bmatrix} & \in \mathcal{S}^{3,\text{vec}}_+ \end{array} \tag{6.26}$$

Vectorization of the LMI is performed as explained in Sec. 15.11.

Setting up the linear part

The linear parts (objective, constraints, variables) and the semidefinite terms in the linear expressions are defined exactly as shown in the previous examples.

Setting up the affine conic constraints with semidefinite terms

To define the affine conic constraints, we first set up the affine expressions. The F matrix and the g vector are defined as usual. Additionally, we specify the coefficients for the semidefinite variables. The semidefinite coefficients shown in (6.26) are setup using the function *Task.putafebarfblocktriplet*.

```
        task.putafebarfblocktriplet(barfi, barfj, barfk, barfl, barfv)
```

These affine expressions are then included in their corresponding domains to construct the affine conic constraints. Lastly, the ACCs are appended to the task.

```
        # Append R+ domain and the corresponding ACC
        task.appendacc(task.appendrplusdomain(1), [0], None)
        # Append SVEC_PSD domain and the corresponding ACC
        task.appendacc(task.appendsvecpsdconedomain(3), [1,2,3], None)
```

Source code

Listing 6.12: Source code solving problem (6.25).

```
import sys
from numpy import sqrt
import mosek

# Since the value of infinity is ignored, we define it solely
# for symbolic purposes
inf = 0.0

# Define a stream printer to grab output from MOSEK
def streamprinter(text):
    sys.stdout.write(text)
    sys.stdout.flush()


def main():
    # Create a task object and attach log stream printer
    with mosek.Task() as task:
        task.set_Stream(mosek.streamtype.log, streamprinter)

        # Below is the sparse triplet representation of the F matrix.
        afeidx = [0, 0, 1, 2, 2, 3]
        varidx = [0, 1, 1, 0, 1, 0]
        f_val  = [-1, -1, 3, sqrt(2), sqrt(2), 3]
        g      = [0, -1, 0, -1]

        barcj = [0, 0]
        barck = [0, 1]
        barcl = [0, 1]
        barcv = [1, 1]

        barfi = [0,0]
        barfj = [0,0]
        barfk = [0,1]
```

(continues on next page)

(continued from previous page)

```
        barfl = [0,0]
        barfv = [0,1]

        numvar = 2
        numafe = 4
        BARVARDIM = [2]

        # Append 'numvar' variables.
        # The variables will initially be fixed at zero (x=0).
        task.appendvars(numvar)

        # Append 'numafe' empty affine expressions.
        task.appendafes(numafe)

        # Append matrix variables of sizes in 'BARVARDIM'.
        # The variables will initially be fixed at zero.
        task.appendbarvars(BARVARDIM)

        # Set the linear terms in the objective.
        task.putcj(0, 1.0)
        task.putcj(1, 1.0)
        task.putcfix(1.0)
        task.putbarcblocktriplet(barcj, barck, barcl, barcv)

        for j in range(numvar):
            # Set the bounds on variable j
            # blx[j] <= x_j <= bux[j]
            task.putvarbound(j, mosek.boundkey.fr, -inf, +inf)

        # Set up the F matrix of the problem
        task.putafefentrylist(afeidx, varidx, f_val)
        # Set up the g vector of the problem
        task.putafegslice(0, numafe, g)
        task.putafebarfblocktriplet(barfi, barfj, barfk, barfl, barfv)

        # Append R+ domain and the corresponding ACC
        task.appendacc(task.appendrplusdomain(1), [0], None)
        # Append SVEC_PSD domain and the corresponding ACC
        task.appendacc(task.appendsvecpsdconedomain(3), [1,2,3], None)

        # Input the objective sense (minimize/maximize)
        task.putobjsense(mosek.objsense.minimize)

        # Solve the problem and print summary
        task.optimize()
        task.solutionsummary(mosek.streamtype.msg)

        # Get status information about the solution
        prosta = task.getprosta(mosek.soltype.itr)
        solsta = task.getsolsta(mosek.soltype.itr)

        if (solsta == mosek.solsta.optimal):
            xx = task.getxx(mosek.soltype.itr)
            barx = task.getbarxj(mosek.soltype.itr, 0)

            print("Optimal solution:\nx=%s\nbarx=%s" % (xx, barx))
```

(continues on next page)

(continued from previous page)

```
        elif (solsta == mosek.solsta.dual_infeas_cer or
              solsta == mosek.solsta.prim_infeas_cer):
            print("Primal or dual infeasibility certificate found.\n")
        elif solsta == mosek.solsta.unknown:
            print("Unknown solution status")
        else:
            print("Other solution status")


# call the main function
try:
    main()
except mosek.MosekException as e:
    print("ERROR: %s" % str(e.errno))
    if e.msg is not None:
        print("\t%s" % e.msg)
        sys.exit(1)
except:
    import traceback
    traceback.print_exc()
    sys.exit(1)
```

6.8 Integer Optimization

An optimization problem where one or more of the variables are constrained to integer values is called a (mixed) integer optimization problem. **MOSEK** supports integer variables in combination with linear, quadratic and quadratically constrtained and conic problems (except semidefinite). See the previous tutorials for an introduction to how to model these types of problems.

6.8.1 Example MILO1

We use the example

$$\begin{array}{lrcl} \text{maximize} & x_0 + 0.64x_1 & & \\ \text{subject to} & 50x_0 + 31x_1 & \leq & 250, \\ & 3x_0 - 2x_1 & \geq & -4, \\ & x_0, x_1 \geq 0 & & \text{and integer} \end{array} \tag{6.27}$$

to demonstrate how to set up and solve a problem with integer variables. It has the structure of a linear optimization problem except for integrality constraints on the variables. Therefore, only the specification of the integer constraints requires something new compared to the linear optimization problem discussed previously.

First, the integrality constraints are imposed using the function *Task.putvartype* or one of its bulk analogues:

```
        task.putvartypelist([0, 1],
                            [mosek.variabletype.type_int,
                             mosek.variabletype.type_int])
```

Next, the example demonstrates how to set various useful parameters of the mixed-integer optimizer. See Sec. 13.4 for details.

```
        # Set max solution time
        task.putdouparam(mosek.dparam.mio_max_time, 60.0);
```

The complete source for the example is listed Listing 6.13. Please note that when we fetch the solution then the integer solution is requested by using *soltype.itg*. No dual solution is defined for integer optimization problems.

Listing 6.13: Source code implementing problem (6.27).

```
import sys
import mosek

# Since the actual value of Infinity is ignores, we define it solely
# for symbolic purposes:
inf = 0.0

# Define a stream printer to grab output from MOSEK
def streamprinter(text):
    sys.stdout.write(text)
    sys.stdout.flush()


def main():
    # Create a task
    with mosek.Task() as task:
        # Attach a printer to the task
        task.set_Stream(mosek.streamtype.log, streamprinter)

        bkc = [mosek.boundkey.up, mosek.boundkey.lo]
        blc = [-inf, -4.0]
        buc = [250.0, inf]

        bkx = [mosek.boundkey.lo, mosek.boundkey.lo]
        blx = [0.0, 0.0]
        bux = [inf, inf]

        c = [1.0, 0.64]

        asub = [[0, 1], [0, 1]]
        aval = [[50.0, 3.0], [31.0, -2.0]]

        numvar = len(bkx)
        numcon = len(bkc)

        # Append 'numcon' empty constraints.
        # The constraints will initially have no bounds.
        task.appendcons(numcon)

        #Append 'numvar' variables.
        # The variables will initially be fixed at zero (x=0).
        task.appendvars(numvar)

        for j in range(numvar):
            # Set the linear term c_j in the objective.
            task.putcj(j, c[j])
            # Set the bounds on variable j
            # blx[j] <= x_j <= bux[j]
            task.putvarbound(j, bkx[j], blx[j], bux[j])
            # Input column j of A
            task.putacol(j,                  # Variable (column) index.
                         # Row index of non-zeros in column j.
                         asub[j],
                         aval[j])            # Non-zero Values of column j.
```

(continues on next page)

(continued from previous page)

```
        task.putconboundlist(range(numcon), bkc, blc, buc)

        # Input the objective sense (minimize/maximize)
        task.putobjsense(mosek.objsense.maximize)

        # Define variables to be integers
        task.putvartypelist([0, 1],
                            [mosek.variabletype.type_int,
                             mosek.variabletype.type_int])

        # Set max solution time
        task.putdouparam(mosek.dparam.mio_max_time, 60.0);

        # Optimize the task
        task.optimize()
        task.writedata("milo1.ptf")

        # Print a summary containing information
        # about the solution for debugging purposes
        task.solutionsummary(mosek.streamtype.msg)

        prosta = task.getprosta(mosek.soltype.itg)
        solsta = task.getsolsta(mosek.soltype.itg)

        # Output a solution
        xx = task.getxx(mosek.soltype.itg)

        if solsta in [mosek.solsta.integer_optimal]:
            print("Optimal solution: %s" % xx)
        elif solsta == mosek.solsta.prim_feas:
            print("Feasible solution: %s" % xx)
        elif mosek.solsta.unknown:
            if prosta == mosek.prosta.prim_infeas_or_unbounded:
                print("Problem status Infeasible or unbounded.\n")
            elif prosta == mosek.prosta.prim_infeas:
                print("Problem status Infeasible.\n")
            elif prosta == mosek.prosta.unkown:
                print("Problem status unkown.\n")
            else:
                print("Other problem status.\n")
        else:
            print("Other solution status")


# call the main function
try:
    main()
except mosek.MosekException as msg:
    #print "ERROR: %s" % str(code)
    if msg is not None:
        print("\t%s" % msg)
        sys.exit(1)
except:
    import traceback
    traceback.print_exc()
    sys.exit(1)
```

6.8.2 Specifying an initial solution

It is a common strategy to provide a starting feasible point (if one is known in advance) to the mixed-integer solver. This can in many cases reduce solution time.

There are two modes for **MOSEK** to utilize an initial solution.

- **A complete solution. MOSEK** will first try to check if the current value of the primal variable solution is a feasible point. The solution can either come from a previous solver call or can be entered by the user, however the full solution with values for all variables (both integer and continuous) must be provided. This check is always performed and does not require any extra action from the user. The outcome of this process can be inspected via information items *iinfitem.mio_initial_feasible_solution* and *dinfitem.mio_initial_feasible_solution_obj*, and via the `Initial feasible solution objective` entry in the log.
- **A partial integer solution. MOSEK** can also try to construct a feasible solution by fixing integer variables to the values provided by the user (rounding if necessary) and optimizing over the remaining continuous variables. In this setup the user must provide initial values for all integer variables. This action is only performed if the parameter *iparam.mio_construct_sol* is switched on. The outcome of this process can be inspected via information items *iinfitem.mio_construct_solution* and *dinfitem.mio_construct_solution_obj*, and via the `Construct solution objective` entry in the log.

In the following example we focus on inputting a partial integer solution.

$$\begin{array}{ll} \text{maximize} & 7x_0 + 10x_1 + x_2 + 5x_3 \\ \text{subject to} & x_0 + x_1 + x_2 + x_3 \leq 2.5 \\ & x_0, x_1, x_2 \in \mathbb{Z} \\ & x_0, x_1, x_2, x_3 \geq 0 \end{array} \tag{6.28}$$

Solution values can be set using *Task.putxx*, *Task.putxxslice* or similar .

Listing 6.14: Implementation of problem (6.28) specifying an initial solution.

```
        # Assign values to integer variables.
        # (We only set a slice of xx)
        task.putxxslice(mosek.soltype.itg, 0, 3, [1.0, 1.0, 0.0])

        # Request constructing the solution from integer variable values
        task.putintparam(mosek.iparam.mio_construct_sol, mosek.onoffkey.on)
```

The log output from the optimizer will in this case indicate that the inputted values were used to construct an initial feasible solution:

```
Construct solution objective       : 1.950000000000e+01
```

The same information can be obtained from the API:

Listing 6.15: Retrieving information about usage of initial solution

```
            constr = task.getintinf(mosek.iinfitem.mio_construct_solution)
            constrVal = task.getdouinf(mosek.dinfitem.mio_construct_solution_obj)
            print("Construct solution utilization: {0}\nConstruct solution objective:
↪{1:.3f}\n".format(constr, constrVal))
```

6.8.3 Example MICO1

Integer variables can also be used arbitrarily in conic problems (except semidefinite). We refer to the previous tutorials for how to set up a conic optimization problem. Here we present sample code that sets up a simple optimization problem:

$$\begin{array}{ll} \text{minimize} & x^2 + y^2 \\ \text{subject to} & x \geq e^y + 3.8, \\ & x, y \text{ integer}. \end{array} \tag{6.29}$$

The canonical conic formulation of (6.29) suitable for Optimizer API for Python is

$$\begin{array}{lll} \text{minimize} & t & \\ \text{subject to} & (t, x, y) \in \mathcal{Q}^3 & (t \geq \sqrt{x^2 + y^2}) \\ & (x - 3.8, 1, y) \in K_{\exp} & (x - 3.8 \geq e^y) \\ & x, y \text{ integer}, & \\ & t \in \mathbb{R}. & \end{array} \tag{6.30}$$

Listing 6.16: Implementation of problem (6.30).

```
with mosek.Task() as task:
    task.set_Stream(mosek.streamtype.log, streamprinter)

    task.appendvars(3)    # x, y, t
    x, y, t = 0, 1, 2
    task.putvarboundsliceconst(0, 3, mosek.boundkey.fr, -0.0, 0.0)

    # Integrality constraints
    task.putvartypelist([x,y], [mosek.variabletype.type_int]*2)

    # Set up the affine expression
    # x, x-3.8, y, t, 1.0
    task.appendafes(5)
    task.putafefentrylist([0, 1, 2, 3],
                          [x,x,y,t],
                          [1,1,1,1])
    task.putafegslice(0, 5, [0, -3.8, 0, 0, 1.0])

    # Add constraint (x-3.8, 1, y) \in \EXP
    task.appendacc(task.appendprimalexpconedomain(), [1, 4, 2], None)

    # Add constraint (t, x, y) \in \QUAD
    task.appendacc(task.appendquadraticconedomain(3), [3, 0, 2], None)

    # Objective
    task.putobjsense(mosek.objsense.minimize)
    task.putcj(t, 1)

    # Optimize the task
    task.optimize()
```

(continues on next page)

(continued from previous page)

```
    task.solutionsummary(mosek.streamtype.msg)

    xx = task.getxxslice(mosek.soltype.itg, 0, 2)
    print(xx)
```

Error and solution status handling were omitted for readability.

6.9 Disjunctive constraints

A **disjunctive constraint (DJC)** involves of a number of affine conditions combined with the logical operators or ($\vee$) and optionally and ($\wedge$) into a formula in *disjunctive normal form*, that is a disjunction of conjunctions. Specifically, a disjunctive constraint has the form of a disjunction

$$T_1 \text{ or } T_2 \text{ or} \cdots \text{ or } T_t \tag{6.31}$$

where each T_i is written as a conjunction

$$T_i = T_{i,1} \text{ and } T_{i,2} \text{ and} \cdots \text{ and } T_{i,s_i} \tag{6.32}$$

and each $T_{i,j}$ is an affine condition (affine equation or affine inequality) of the form $D_{ij}x + d_{ij} \in \mathcal{D}_{ij}$ with $\mathcal{D}_{ij}$ being one of the affine domains from Sec. 15.11.1. A disjunctive constraint (DJC) can therefore be succinctly written as

$$\bigvee_{i=1}^{t} \bigwedge_{j=1}^{s_i} T_{i,j} \tag{6.33}$$

where each $T_{i,j}$ is an affine condition.

Each T_i is called a **term** or **clause** of the disjunctive constraint and t is the number of terms. Each condition $T_{i,j}$ is called a **simple term** and s_i is called the **size** of the i-th term.

A disjunctive constraint is satisfied if at least one of its terms (clauses) is satisfied. A term (clause) is satisfied if all of its constituent simple terms are satisfied. A problem containing DJCs will be solved by the mixed-integer optimizer.

Note that nonlinear cones are not allowed as one of the domains $\mathcal{D}_{ij}$ inside a DJC.

6.9.1 Applications

Disjunctive constraints are a convenient and expressive syntactical tool. Then can be used to phrase many constructions appearing especially in mixed-integer modelling. Here are some examples.

- **Complementarity.** The condition $xy = 0$, where x, y are scalar variables, is equivalent to

  $$x = 0 \text{ or } y = 0.$$

  It is a DJC with two terms, each of size 1.

- **Semicontinuous variable.** A semicontinuous variable is a scalar variable which takes values in $\{0\} \cup [a, +\infty]$. This can be expressed as

  $$x = 0 \text{ or } x \geq a.$$

  It is again a DJC with two terms, each of size 1.

- **Exact absolute value.** The constraint $t = |x|$ is not convex, but can be written as

  $$(x \geq 0 \text{ and } t = x) \text{ or } (x \leq 0 \text{ and } t = -x)$$

  It is a DJC with two terms, each of size 2.

- **Indicator.** Suppose z is a Boolean variable. Then we can write the indicator constraint $z = 1 \implies a^T x \leq b$ as

$$(z = 1 \text{ and } a^T x \leq b) \text{ or } (z = 0)$$

  which is a DJC with two terms, of sizes, respectively, 2 and 1.

- **Piecewise linear functions.** Suppose $a_1 \leq \cdots \leq a_{k+1}$ and $f : [a_1, a_{k+1}] \to \mathbb{R}$ is a piecewise linear function, given on the i-th of k intervals $[a_i, a_{i+1}]$ by a different affine expression $f_i(x)$. Then we can write the constraint $y = f(x)$ as

$$\bigvee_{i=1}^{k} (a_i \leq y \text{ and } y \leq a_{i+1} \text{ and } y - f_i(x) = 0)$$

  making it a DJC with k terms, each of size 3.

On the other hand most DJCs are equivalent to a mixed-integer linear program through a big-M reformulation. In some cases, when a suitable big-M is known to the user, writing such a formulation directly may be more efficient than formulating the problem as a DJC. See Sec. 13.4.5 for a discussion of this topic.

Disjunctive constraints can be added to any problem which includes linear constraints, affine conic constraints (without semidefinite domains) or integer variables.

6.9.2 Example DJC1

In this tutorial we will consider the following sample demonstration problem:

$$\begin{array}{ll} \text{minimize} & 2x_0 + x_1 + 3x_2 + x_3 \\ \text{subject to} & x_0 + x_1 + x_2 + x_3 \geq -10, \\ & \left(\begin{array}{c} x_0 - 2x_1 \leq -1 \\ \text{and} \\ x_2 = x_3 = 0 \end{array} \right) \text{ or } \left(\begin{array}{c} x_2 - 3x_3 \leq -2 \\ \text{and} \\ x_0 = x_1 = 0 \end{array} \right), \\ & x_i = 2.5 \text{ for at least one } i \in \{0, 1, 2, 3\}. \end{array} \tag{6.34}$$

The problem has two DJCs: the first one has 2 terms. The second one, which we can write as $\bigvee_{i=0}^{3}(x_i = 2.5)$, has 4 terms (clauses).

We begin by expressing problem (6.34) in the format where all simple terms are of the form $D_{ij}x+d_{ij} \in \mathcal{D}_{ij}$, that is of the form *a sequence of affine expressions belongs to a linear domain*:

$$\begin{array}{ll} \text{minimize} & 2x_0 + x_1 + 3x_2 + x_3 \\ \text{subject to} & x_0 + x_1 + x_2 + x_3 \geq -10, \\ & \left(\begin{array}{c} x_0 - 2x_1 + 1 \in \mathbb{R}^1_{\leq 0} \\ \text{and} \\ (x_2, x_3) \in 0^2 \end{array} \right) \text{ or } \left(\begin{array}{c} x_2 - 3x_3 + 2 \in \mathbb{R}^1_{\leq 0} \\ \text{and} \\ (x_0, x_1) \in 0^2 \end{array} \right), \\ & (x_0 - 2.5 \in 0^1) \text{ or } (x_1 - 2.5 \in 0^1) \text{ or } (x_2 - 2.5 \in 0^1) \text{ or } (x_3 - 2.5 \in 0^1), \end{array} \tag{6.35}$$

where 0^n denotes the n-dimensional zero domain and $\mathbb{R}^n_{\leq 0}$ denotes the n-dimensional nonpositive orthant, as in Sec. 15.11.

Now we show how to add the two DJCs from (6.35). This involves three steps:

- storing the affine expressions which appear in the DJCs,
- creating the required domains, and
- combining the two into the description of the DJCs.

Readers familiar with Sec. 6.2 will find that the process is completely analogous to the process of adding affine conic constraints (ACCs). In fact we would recommend Sec. 6.2 as a means of familiarizing with the structures used here at a slightly lower level of complexity.

6.9.3 Step 1: add affine expressions

In the first step we need to store all affine expressions appearing in the problem, that is the rows of the expressions $D_{ij}x + d_{ij}$. In problem (6.35) the disjunctive constraints contain altogether the following affine expressions:

$$\begin{array}{ll} (0) & x_0 - 2x_1 + 1 \\ (1) & x_2 - 3x_3 + 2 \\ (2) & x_0 \\ (3) & x_1 \\ (4) & x_2 \\ (5) & x_3 \\ (6) & x_0 - 2.5 \\ (7) & x_1 - 2.5 \\ (8) & x_2 - 2.5 \\ (9) & x_3 - 2.5 \end{array} \tag{6.36}$$

To store affine expressions (**AFE** for short) **MOSEK** provides a matrix $\mathbf{F}$ and a vector $\mathbf{g}$ with the understanding that every row of

$$\mathbf{F}x + \mathbf{g}$$

defines one affine expression. The API functions with infix `afe` are used to operate on $\mathbf{F}$ and $\mathbf{g}$, add rows, add columns, set individual elements, set blocks etc. similarly to the methods for operating on the A matrix of linear constraints. The storage matrix $\mathbf{F}$ is a sparse matrix, therefore only nonzero elements have to be explicitly added.

Remark: the storage $\mathbf{F}, \mathbf{g}$ may, but does not have to be, kept in the same order in which the expressions enter DJCs. In fact in (6.36) we have chosen to list the linear expressions in a different, convenient order. It is also possible to store some expressions only once if they appear multiple times in DJCs.

Given the list (6.36), we initialize the AFE storage as (only nonzeros are listed and for convenience we list the content of (6.36) alongside in the leftmost column):

$$\begin{array}{ll} (0) & x_0 - 2x_1 + 1 \\ (1) & x_2 - 3x_3 + 2 \\ (2) & x_0 \\ (3) & x_1 \\ (4) & x_2 \\ (5) & x_3 \\ (6) & x_0 - 2.5 \\ (7) & x_1 - 2.5 \\ (8) & x_2 - 2.5 \\ (9) & x_3 - 2.5 \end{array} \quad \mathbf{F} = \begin{bmatrix} 1 & -2 & & \\ & & 1 & -3 \\ 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \\ 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{bmatrix}, \quad \mathbf{g} = \begin{bmatrix} 1 \\ 2 \\ \\ \\ \\ \\ -2.5 \\ -2.5 \\ -2.5 \\ -2.5 \end{bmatrix}. \tag{6.37}$$

Initially $\mathbf{F}$ and $\mathbf{g}$ are empty (have 0 rows). We construct them as follows. First, we append a number of empty rows:

```
numafe = 10
task.appendafes(numafe)
```

We now have $\mathbf{F}$ and $\mathbf{g}$ with 10 rows of zeros and we fill them up to obtain (6.37).

```
fafeidx = [0, 0, 1, 1, 2, 3, 4, 5, 6, 7, 8, 9]
fvaridx = [0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3]
fval    = [1.0, -2.0, 1.0, -3.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
g       = [1.0, 2.0, 0.0, 0.0, 0.0, 0.0, -2.5, -2.5, -2.5, -2.5]

task.putafefentrylist(fafeidx, fvaridx, fval)
task.putafegslice(0, numafe, g)
```

We have now created the matrices from (6.37). Note that at this point we have *not defined any DJCs yet*. All we did was define some affine expressions and place them in a generic AFE storage facility to be used later.

6.9.4 Step 2: create domains

Next, we create all the domains $\mathcal{D}_{ij}$ appearing in all the simple terms of all DJCs. Domains are created with functions with infix `domain`. In the case of (6.35) there are three different domains appearing:

$$0^1,\ 0^2,\ \mathbb{R}^1_{\leq 0}.$$

We create them with the corresponding functions:

```
zero1   = task.appendrzerodomain(1)
zero2   = task.appendrzerodomain(2)
rminus1 = task.appendrminusdomain(1)
```

The function returns a domain index, which is just the position in the list of all domains (potentially) created for the problem. At this point the domains are just stored in the list of domains, but not yet used for anything.

6.9.5 Step 3: create the actual disjunctive constraints

We are now in position to create the disjunctive constraints. DJCs are created with functions with infix `djc`. The function *`Task.appenddjcs`* will append a number of initially empty DJCs to the task:

```
numdjc = 2
task.appenddjcs(numdjc)
```

We can then define each disjunction with the method *`Task.putdjc`*. It will require the following data:

- the list `termsizelist` of the sizes of all terms of the DJC,
- the list `afeidxlist` of indices of AFEs to be used in the constraint. These are the row numbers in $\mathbf{F}, \mathbf{g}$ which contain the required affine expressions.
- the list `domidxlist` of the domains for all the simple terms.

For example, consider the first DJC of (6.35). Below we format this DJC by replacing each affine expression with the index of that expression in (6.37) and each domain with its index we obtained in Step 2:

$$\begin{array}{ccc} \left(x_0 - 2x_1 + 1 \in \mathbb{R}^1_{\leq 0} \text{ and } (x_2, x_3) \in 0^2\right) & \text{or} & \left(x_2 - 3x_3 + 2 \in \mathbb{R}^1_{\leq 0} \text{ and } (x_0, x_1) \in 0^2\right) \\ \underbrace{((0) \in \texttt{rminus1} \text{ and } ((4),(5)) \in \texttt{zero2})}_{\texttt{term of size 2}} & \text{or} & \underbrace{((1) \in \texttt{rminus1} \text{ and } ((2),(3)) \in \texttt{zero2})}_{\texttt{term of size 2}} \end{array} \tag{6.38}$$

It implies that the DJC will be represented by the following data:

- `termsizelist = [2, 2]`,
- `afeidxlist = [0, 4, 5, 1, 2, 3]`,
- `domidxlist = [rminus1, zero2, rminus1, zero2]`.

The code adding this DJC will therefore look as follows:

```
task.putdjc(0,                                          # DJC index
            [rminus1, zero2, rminus1, zero2],           # Domains
↪(domidxlist)
            [0, 4, 5, 1, 2, 3],                         # AFE indices
↪(afeidxlist)
            None,                                       # Unused
            [2, 2] )                                    # Term sizes
↪(termsizelist)
```

Note that number of AFEs used in `afeidxlist` must match the sum of dimensions of all the domains (here: $6 == 1+2+1+2$) and the number of domains must match the sum of all term sizes (here: $4 == 2+2$).

For similar reasons the second DJC of problem (6.35) will have the description:

$$\begin{array}{cccccccc} x_0 - 2.5 \in 0^1 & \text{or} & x_1 - 2.5 \in 0^1 & \text{or} & x_2 - 2.5 \in 0^1 & \text{or} & x_3 - 2.5 \in 0^1 \\ \underbrace{(6) \in \texttt{zero1}}_{\texttt{term of size 1}} & \text{or} & \underbrace{(7) \in \texttt{zero1}}_{\texttt{term of size 1}} & \text{or} & \underbrace{(8) \in \texttt{zero1}}_{\texttt{term of size 1}} & \text{or} & \underbrace{(9) \in \texttt{zero1}}_{\texttt{term of size 1}} \end{array} \tag{6.39}$$

- `termsizelist = [1, 1, 1, 1]`,
- `afeidxlist = [6, 7, 8, 9]`,
- `domidxlist = [zero1, zero1, zero1, zero1]`.

```
        task.putdjc(1,                                          # DJC index
                    [zero1, zero1, zero1, zero1],               # Domains
↪(domidxlist)
                    [6, 7, 8, 9],                               # AFE indices
↪(afeidxlist)
                    None,                                       # Unused
                    [1, 1, 1, 1] )                              # Term sizes
↪(termidxlist)
```

This completes the setup of the disjunctive constraints.

6.9.6 Example DJC1 full code

We refer to Sec. 6.1 for instructions how to initialize a **MOSEK** session, add variables and set up the objective and linear constraints. All else that remains is to call the solver with *Task.optimize* and retrieve the solution with *Task.getxx*. Since our problem contains a DJC, and thus is solved by the mixed-integer optimizer, we fetch the integer solution. The full code solving problem (6.34) is shown below.

Listing 6.17: Full code of example DJC1.

```
import sys
from mosek import *

# Since the value of infinity is ignored, we define it solely
# for symbolic purposes
inf = 0.0

def main():
    # Create a task object
    with Task() as task:
        # Append free variables
        numvar = 4
        task.appendvars(numvar)
        task.putvarboundsliceconst(0, numvar, boundkey.fr, -inf, inf)

        # The linear part: the linear constraint
        task.appendcons(1)
        task.putarow(0, range(numvar), [1] * numvar)
        task.putconbound(0, boundkey.lo, -10.0, -10.0)

        # The linear part: objective
        task.putobjsense(objsense.minimize)
        task.putclist(range(numvar), [2, 1, 3, 1])
```

(continues on next page)

(continued from previous page)

```
        # Fill in the affine expression storage F, g
        numafe = 10
        task.appendafes(numafe)

        fafeidx = [0, 0, 1, 1, 2, 3, 4, 5, 6, 7, 8, 9]
        fvaridx = [0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3]
        fval    = [1.0, -2.0, 1.0, -3.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
        g       = [1.0, 2.0, 0.0, 0.0, 0.0, 0.0, -2.5, -2.5, -2.5, -2.5]

        task.putafefentrylist(fafeidx, fvaridx, fval)
        task.putafegslice(0, numafe, g)

        # Create domains
        zero1   = task.appendrzerodomain(1)
        zero2   = task.appendrzerodomain(2)
        rminus1 = task.appendrminusdomain(1)

        # Append disjunctive constraints
        numdjc = 2
        task.appenddjcs(numdjc)

        # First disjunctive constraint
        task.putdjc(0,                                          # DJC index
                    [rminus1, zero2, rminus1, zero2],           # Domains
↪(domidxlist)
                    [0, 4, 5, 1, 2, 3],                         # AFE indices
↪(afeidxlist)
                    None,                                       # Unused
                    [2, 2] )                                    # Term sizes
↪(termsizelist)

        # Second disjunctive constraint
        task.putdjc(1,                                          # DJC index
                    [zero1, zero1, zero1, zero1],               # Domains
↪(domidxlist)
                    [6, 7, 8, 9],                               # AFE indices
↪(afeidxlist)
                    None,                                       # Unused
                    [1, 1, 1, 1] )                              # Term sizes
↪(termidxlist)

        # Useful for debugging
        task.writedata("djc.ptf")                           # Write file in human-
↪readable format
        task.set_Stream(streamtype.log, sys.stdout.write)   # Attach a log stream
↪printer to the task

        # Solve the problem
        task.optimize()

        # Print a summary containing information
        # about the solution for debugging purposes
        task.solutionsummary(streamtype.msg)

        # Get status information about the solution
```

(continues on next page)

(continued from previous page)

```
        sta = task.getsolsta(soltype.itg)

        if (sta == solsta.integer_optimal):
            xx = task.getxx(soltype.itg)

            print("Optimal solution: ")
            for i in range(numvar):
                print("x[" + str(i) + "]=" + str(xx[i]))
        else:
            print("Another solution status")
```

The answer is

```
[0, 0, -12.5, 2.5]
```

6.9.7 Summary and extensions

In this section we presented the most basic usage of the affine expression storage $\mathbf{F}, \mathbf{g}$ to input *affine expressions* used together with *domains* to create *disjunctive constraints* (DJC). Now we briefly point out additional features of his interface which can be useful in some situations for more demanding users. They will be demonstrated in various examples in other tutorials and case studies in this manual.

- It is important to remember that $\mathbf{F}, \mathbf{g}$ has *only a storage function* and during the DJC construction we can pick an arbitrary list of row indices and place them in a domain. It means for example that:
  - It is not necessary to store the AFEs in the same order they will appear in DJCs.
  - The same AFE index can appear more than once in one and/or more conic constraints (this can be used to reduce storage if the same affine expression is used in multiple DJCs).
  - The $\mathbf{F}, \mathbf{g}$ storage can even include rows that are not presently used in any DJC.
- Domains can be reused: multiple DJCs can use the same domain. On the other hand the same type of domain can appear under many `domidx` positions. In this sense the list of created domains also plays only a *storage role*: the domains are only used when they enter a DJC.
- The same affine expression storage $\mathbf{F}, \mathbf{g}$ is shared between disjunctive constraints and affine conic constraints (ACCs, see Sec. 6.2).
- When defining an DJC an additional constant vector b can be provided to modify the constant terms coming from $\mathbf{g}$ but only for this particular DJC. This could be useful to reduce $\mathbf{F}$ storage space if, for example, many expressions $D^T x - b_i$ with the same linear part $D^T x$, but varying constant terms b_i, are to be used throughout DJCs.

6.10 Quadratic Optimization

MOSEK can solve quadratic and quadratically constrained problems, as long as they are convex. This class of problems can be formulated as follows:

$$\begin{array}{lrcccll} \text{minimize} & & & \frac{1}{2}x^T Q^o x + c^T x + c^f & & & \\ \text{subject to} & l_k^c & \leq & \frac{1}{2}x^T Q^k x + \sum_{j=0}^{n-1} a_{k,j} x_j & \leq & u_k^c, & k = 0, \ldots, m-1, \\ & l_j^x & \leq & x_j & \leq & u_j^x, & j = 0, \ldots, n-1. \end{array} \tag{6.40}$$

Without loss of generality it is assumed that Q^o and Q^k are all symmetric because

$$x^T Q x = \frac{1}{2} x^T (Q + Q^T) x.$$

This implies that a non-symmetric Q can be replaced by the symmetric matrix $\frac{1}{2}(Q + Q^T)$.

The problem is required to be convex. More precisely, the matrix Q^o must be positive semi-definite and the kth constraint must be of the form

$$l_k^c \leq \frac{1}{2} x^T Q^k x + \sum_{j=0}^{n-1} a_{k,j} x_j \tag{6.41}$$

with a negative semi-definite Q^k or of the form

$$\frac{1}{2} x^T Q^k x + \sum_{j=0}^{n-1} a_{k,j} x_j \leq u_k^c.$$

with a positive semi-definite Q^k. This implies that quadratic equalities are *not* allowed. Specifying a non-convex problem will result in an error when the optimizer is called.

A matrix is positive semidefinite if all the eigenvalues of Q are nonnegative. An alternative statement of the positive semidefinite requirement is

$$x^T Q x \geq 0, \quad \forall x.$$

If the convexity (i.e. semidefiniteness) conditions are not met **MOSEK** will not produce reliable results or work at all.

6.10.1 Example: Quadratic Objective

We look at a small problem with linear constraints and quadratic objective:

$$\begin{array}{lll} \text{minimize} & & x_1^2 + 0.1 x_2^2 + x_3^2 - x_1 x_3 - x_2 \\ \text{subject to} & 1 \leq & x_1 + x_2 + x_3 \\ & 0 \leq & x. \end{array} \tag{6.42}$$

The matrix formulation of (6.42) has:

$$Q^o = \begin{bmatrix} 2 & 0 & -1 \\ 0 & 0.2 & 0 \\ -1 & 0 & 2 \end{bmatrix}, c = \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix}, A = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix},$$

with the bounds:

$$l^c = 1, u^c = \infty, l^x = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \text{ and } u^x = \begin{bmatrix} \infty \\ \infty \\ \infty \end{bmatrix}$$

Please note the explicit $\frac{1}{2}$ in the objective function of (6.40) which implies that diagonal elements must be doubled in Q, i.e. $Q_{11} = 2$ even though 1 is the coefficient in front of x_1^2 in (6.42).

Setting up the linear part

The linear parts (constraints, variables, objective) are set up using exactly the same methods as for linear problems, and we refer to Sec. 6.1 for all the details. The same applies to technical aspects such as defining an optimization task, retrieving the solution and so on.

Setting up the quadratic objective

The quadratic objective is specified using the function *Task.putqobj*. Since Q^o is symmetric only the lower triangular part of Q^o is inputted. In fact entries from above the diagonal may *not* appear in the input.

The lower triangular part of the matrix Q^o is specified using an unordered sparse triplet format (for details, see Sec. 15.1.4):

```
qsubi = [0, 1, 2, 2]
qsubj = [0, 1, 0, 2]
qval = [2.0, 0.2, -1.0, 2.0]
```

Please note that

- only non-zero elements are specified (any element not specified is 0 by definition),
- the order of the non-zero elements is insignificant, and
- *only* the lower triangular part should be specified.

Finally, this definition of Q^o is loaded into the task:

```
        task.putqobj(qsubi, qsubj, qval)
```

Source code

Listing 6.18: Source code implementing problem (6.42).

```
import sys, os, mosek

# Since the actual value of Infinity is ignored, we define it solely
# for symbolic purposes:
inf = 0.0

# Define a stream printer to grab output from MOSEK
def streamprinter(text):
    sys.stdout.write(text)
    sys.stdout.flush()


def main():
    # Create a task
    with mosek.Task() as task:
        task.set_Stream(mosek.streamtype.log, streamprinter)
        # Set up and input bounds and linear coefficients
        bkc = [mosek.boundkey.lo]
        blc = [1.0]
        buc = [inf]
        numvar = 3
        bkx = [mosek.boundkey.lo] * numvar
        blx = [0.0] * numvar
        bux = [inf] * numvar
        c = [0.0, -1.0, 0.0]
        asub = [[0], [0], [0]]
        aval = [[1.0], [1.0], [1.0]]

        numvar = len(bkx)
        numcon = len(bkc)

        # Append 'numcon' empty constraints.
        # The constraints will initially have no bounds.
        task.appendcons(numcon)

        # Append 'numvar' variables.
        # The variables will initially be fixed at zero (x=0).
        task.appendvars(numvar)

        for j in range(numvar):
            # Set the linear term c_j in the objective.
            task.putcj(j, c[j])
```

(continues on next page)

(continued from previous page)

```
            # Set the bounds on variable j
            # blx[j] <= x_j <= bux[j]
            task.putvarbound(j, bkx[j], blx[j], bux[j])
            # Input column j of A
            task.putacol(j,                   # Variable (column) index.
                         # Row index of non-zeros in column j.
                         asub[j],
                         aval[j])             # Non-zero Values of column j.
        for i in range(numcon):
            task.putconbound(i, bkc[i], blc[i], buc[i])

        # Set up and input quadratic objective
        qsubi = [0, 1, 2, 2]
        qsubj = [0, 1, 0, 2]
        qval = [2.0, 0.2, -1.0, 2.0]

        task.putqobj(qsubi, qsubj, qval)

        # Input the objective sense (minimize/maximize)
        task.putobjsense(mosek.objsense.minimize)

        # Optimize
        task.optimize()
        # Print a summary containing information
        # about the solution for debugging purposes
        task.solutionsummary(mosek.streamtype.msg)

        prosta = task.getprosta(mosek.soltype.itr)
        solsta = task.getsolsta(mosek.soltype.itr)

        # Output a solution
        xx = task.getxx(mosek.soltype.itr)

        if solsta == mosek.solsta.optimal:
            print("Optimal solution: %s" % xx)
        elif solsta == mosek.solsta.dual_infeas_cer:
            print("Primal or dual infeasibility.\n")
        elif solsta == mosek.solsta.prim_infeas_cer:
            print("Primal or dual infeasibility.\n")
        elif mosek.solsta.unknown:
            print("Unknown solution status")
        else:
            print("Other solution status")


# call the main function
try:
    main()
except mosek.MosekException as e:
    print("ERROR: %s" % str(e.errno))
    if e.msg is not None:
        import traceback
        traceback.print_exc()
        print("\t%s" % e.msg)
    sys.exit(1)
except:
```

(continues on next page)

(continued from previous page)

```
    import traceback
    traceback.print_exc()
    sys.exit(1)
```

6.10.2 Example: Quadratic constraints

In this section we show how to solve a problem with quadratic constraints. Please note that quadratic constraints are subject to the convexity requirement (6.41).

Consider the problem:

$$\begin{array}{lrcc} \text{minimize} & & & x_1^2 + 0.1x_2^2 + x_3^2 - x_1x_3 - x_2 \\ \text{subject to} & 1 & \leq & x_1 + x_2 + x_3 - x_1^2 - x_2^2 - 0.1x_3^2 + 0.2x_1x_3, \\ & & & x \geq 0. \end{array}$$

This is equivalent to

$$\begin{array}{lrcl} \text{minimize} & & & \frac{1}{2}x^T Q^o x + c^T x \\ \text{subject to} & & \frac{1}{2}x^T Q^0 x + Ax & \geq \quad b, \\ & & x \geq 0, & \end{array} \tag{6.43}$$

where

$$Q^o = \begin{bmatrix} 2 & 0 & -1 \\ 0 & 0.2 & 0 \\ -1 & 0 & 2 \end{bmatrix}, c = \begin{bmatrix} 0 & -1 & 0 \end{bmatrix}^T, A = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}, b = 1.$$

$$Q^0 = \begin{bmatrix} -2 & 0 & 0.2 \\ 0 & -2 & 0 \\ 0.2 & 0 & -0.2 \end{bmatrix}.$$

The linear parts and quadratic objective are set up the way described in the previous tutorial.

Setting up quadratic constraints

To add quadratic terms to the constraints we use the function *Task.putqconk*.

```
        qsubi = [0, 1, 2, 2]
        qsubj = [0, 1, 2, 0]
        qval = [-2.0, -2.0, -0.2, 0.2]

        # put Q^0 in constraint with index 0.

        task.putqconk(0, qsubi, qsubj, qval)
```

While *Task.putqconk* adds quadratic terms to a specific constraint, it is also possible to input all quadratic terms in one chunk using the *Task.putqcon* function.

Source code

Listing 6.19: Implementation of the quadratically constrained problem (6.43).

```
import sys
import mosek

# Since the actual value of Infinity is ignores, we define it solely
# for symbolic purposes:
inf = 0.0
```

(continues on next page)

(continued from previous page)

```
# Define a stream printer to grab output from MOSEK
def streamprinter(text):
    sys.stdout.write(text)
    sys.stdout.flush()


def main():
    # Create a task
    with mosek.Task() as task:
        # Attach a printer to the task
        task.set_Stream(mosek.streamtype.log, streamprinter)

        # Set up and input bounds and linear coefficients
        bkc = [mosek.boundkey.lo]
        blc = [1.0]
        buc = [inf]

        bkx = [mosek.boundkey.lo,
               mosek.boundkey.lo,
               mosek.boundkey.lo]
        blx = [0.0, 0.0, 0.0]
        bux = [inf, inf, inf]

        c = [0.0, -1.0, 0.0]

        asub = [[0], [0], [0]]
        aval = [[1.0], [1.0], [1.0]]

        numvar = len(bkx)
        numcon = len(bkc)
        NUMANZ = 3
        # Append 'numcon' empty constraints.
        # The constraints will initially have no bounds.
        task.appendcons(numcon)

        #Append 'numvar' variables.
        # The variables will initially be fixed at zero (x=0).
        task.appendvars(numvar)

        #Optionally add a constant term to the objective.
        task.putcfix(0.0)

        for j in range(numvar):
            # Set the linear term c_j in the objective.
            task.putcj(j, c[j])
            # Set the bounds on variable j
            # blx[j] <= x_j <= bux[j]
            task.putvarbound(j, bkx[j], blx[j], bux[j])
            # Input column j of A
            task.putacol(j,                  # Variable (column) index.
                         # Row index of non-zeros in column j.
                         asub[j],
                         aval[j])            # Non-zero Values of column j.

        for i in range(numcon):
```

(continues on next page)

(continued from previous page)

```
            task.putconbound(i, bkc[i], blc[i], buc[i])

        # Set up and input quadratic objective

        qsubi = [0, 1, 2, 2]
        qsubj = [0, 1, 0, 2]
        qval = [2.0, 0.2, -1.0, 2.0]

        task.putqobj(qsubi, qsubj, qval)

        # The lower triangular part of the Q^0
        # matrix in the first constraint is specified.
        # This corresponds to adding the term
        # - x0^2 - x1^2 - 0.1 x2^2 + 0.2 x0 x2

        qsubi = [0, 1, 2, 2]
        qsubj = [0, 1, 2, 0]
        qval = [-2.0, -2.0, -0.2, 0.2]

        # put Q^0 in constraint with index 0.

        task.putqconk(0, qsubi, qsubj, qval)

        # Input the objective sense (minimize/maximize)
        task.putobjsense(mosek.objsense.minimize)

        # Optimize the task
        task.optimize()

        # Print a summary containing information
        # about the solution for debugging purposes
        task.solutionsummary(mosek.streamtype.msg)

        prosta = task.getprosta(mosek.soltype.itr)
        solsta = task.getsolsta(mosek.soltype.itr)

        # Output a solution
        xx = task.getxx(mosek.soltype.itr)

        if solsta == mosek.solsta.optimal:
            print("Optimal solution: %s" % xx)
        elif solsta == mosek.solsta.dual_infeas_cer:
            print("Primal or dual infeasibility.\n")
        elif solsta == mosek.solsta.prim_infeas_cer:
            print("Primal or dual infeasibility.\n")
        elif mosek.solsta.unknown:
            print("Unknown solution status")
        else:
            print("Other solution status")


# call the main function
try:
    main()
except mosek.MosekException as e:
    print("ERROR: %s" % str(e.errno))
```

(continues on next page)

(continued from previous page)

```
    print("\t%s" % e.msg)
    sys.exit(1)
except:
    import traceback
    traceback.print_exc()
    sys.exit(1)
```

6.11 Problem Modification and Reoptimization

Often one might want to solve not just a single optimization problem, but a sequence of problems, each differing only slightly from the previous one. This section demonstrates how to modify and re-optimize an existing problem.

The example we study is a simple production planning model.

Problem modifications regarding variables, cones, objective function and constraints can be grouped in categories:

- add/remove,
- coefficient modifications,
- bounds modifications.

Especially removing variables and constraints can be costly. Special care must be taken with respect to constraints and variable indexes that may be invalidated.

Depending on the type of modification, **MOSEK** may be able to optimize the modified problem more efficiently exploiting the information and internal state from the previous execution. After optimization, the solution is always stored internally, and is available before next optimization. The former optimal solution may be still feasible, but no longer optimal; or it may remain optimal if the modification of the objective function was small. This special case is discussed in Sec. 14.3.

In general, **MOSEK** exploits dual information and availability of an optimal basis from the previous execution. The simplex optimizer is well suited for exploiting an existing primal or dual feasible solution. Restarting capabilities for interior-point methods are still not as reliable and effective as those for the simplex algorithm. More information can be found in Chapter 10 of the book [Chvatal83].

Parameter settings (see Sec. 7.5) can also be changed between optimizations.

6.11.1 Example: Production Planning

A company manufactures three types of products. Suppose the stages of manufacturing can be split into three parts: Assembly, Polishing and Packing. In the table below we show the time required for each stage as well as the profit associated with each product.

Product no.	Assembly (minutes)	Polishing (minutes)	Packing (minutes)	Profit ($)
0	2	3	2	1.50
1	4	2	3	2.50
2	3	3	2	3.00

With the current resources available, the company has $100,000$ minutes of assembly time, $50,000$ minutes of polishing time and $60,000$ minutes of packing time available per year. We want to know how many items of each product the company should produce each year in order to maximize profit?

Denoting the number of items of each type by x_0, x_1 and x_2, this problem can be formulated as a linear optimization problem:

$$\begin{array}{lrcrcrcl} \text{maximize} & 1.5x_0 & + & 2.5x_1 & + & 3.0x_2 & & \\ \text{subject to} & 2x_0 & + & 4x_1 & + & 3x_2 & \leq & 100000, \\ & 3x_0 & + & 2x_1 & + & 3x_2 & \leq & 50000, \\ & 2x_0 & + & 3x_1 & + & 2x_2 & \leq & 60000, \end{array} \tag{6.44}$$

and

$$x_0, x_1, x_2 \geq 0.$$

Code in Listing 6.20 loads and solves this problem.

Listing 6.20: Setting up and solving problem (6.44)

```
    # Create a task
    with mosek.Task() as task:
        # Bound keys for constraints
        bkc = [mosek.boundkey.up,
               mosek.boundkey.up,
               mosek.boundkey.up]
        # Bound values for constraints
        blc = [-inf, -inf, -inf]
        buc = [100000.0, 50000.0, 60000.0]
        # Bound keys for variables
        bkx = [mosek.boundkey.lo,
               mosek.boundkey.lo,
               mosek.boundkey.lo]
        # Bound values for variables
        blx = [0.0, 0.0, 0.0]
        bux = [+inf, +inf, +inf]
        # Objective coefficients
        csub = [0, 1, 2]
        cval = [1.5, 2.5, 3.0]
        # We input the A matrix column-wise
        # asub contains row indexes
        asub = [0, 1, 2,
                0, 1, 2,
                0, 1, 2]
        # acof contains coefficients
        acof = [2.0, 3.0, 2.0,
                4.0, 2.0, 3.0,
                3.0, 3.0, 2.0]
        # aptrb and aptre contains the offsets into asub and acof where
        # columns start and end respectively
        aptrb = [0, 3, 6]
        aptre = [3, 6, 9]

        numvar = len(bkx)
        numcon = len(bkc)

        # Append the constraints
        task.appendcons(numcon)

        # Append the variables.
        task.appendvars(numvar)

        # Input objective
        task.putcfix(0.0)
        task.putclist(csub, cval)

        # Put constraint bounds
        task.putconboundslice(0, numcon, bkc, blc, buc)

        # Put variable bounds
```

(continues on next page)

(continued from previous page)

```
        task.putvarboundslice(0, numvar, bkx, blx, bux)

        # Input A non-zeros by columns
        for j in range(numvar):
            ptrb, ptre = aptrb[j], aptre[j]
            task.putacol(j,
                         asub[ptrb:ptre],
                         acof[ptrb:ptre])

        # Input the objective sense (minimize/maximize)
        task.putobjsense(mosek.objsense.maximize)

        # Optimize the task
        task.optimize()

        # Output a solution
        xx = task.getsolutionslice(mosek.soltype.bas,
                                   mosek.solitem.xx,
                                   0, numvar)
        print("xx = {}".format(xx))
```

6.11.2 Changing the Linear Constraint Matrix

Suppose we want to change the time required for assembly of product 0 to 3 minutes. This corresponds to setting $a_{0,0} = 3$, which is done by calling the function *Task.putaij* as shown below.

```
        task.putaij(0, 0, 3.0)
```

The problem now has the form:

$$\begin{array}{lrcrcrcl} \text{maximize} & 1.5x_0 & + & 2.5x_1 & + & 3.0x_2 & & \\ \text{subject to} & 3x_0 & + & 4x_1 & + & 3x_2 & \leq & 100000, \\ & 3x_0 & + & 2x_1 & + & 3x_2 & \leq & 50000, \\ & 2x_0 & + & 3x_1 & + & 2x_2 & \leq & 60000, \end{array} \tag{6.45}$$

and

$$x_0, x_1, x_2 \geq 0.$$

After this operation we can reoptimize the problem.

6.11.3 Appending Variables

We now want to add a new product with the following data:

Product no.	Assembly (minutes)	Polishing (minutes)	Packing (minutes)	Profit ($)
3	4	0	1	1.00

This corresponds to creating a new variable x_3, appending a new column to the A matrix and setting a new term in the objective. We do this in Listing 6.21

Listing 6.21: How to add a new variable (column)

```
        ################### Add a new variable ######################
        task.appendvars(1)
        numvar+=1
```

(continues on next page)

(continued from previous page)

```
        # Set bounds on new varaible
        task.putvarbound(task.getnumvar() - 1,
                         mosek.boundkey.lo,
                         0,
                         +inf)

        # Change objective
        task.putcj(task.getnumvar() - 1, 1.0)

        # Put new values in the A matrix
        acolsub = [0, 2]
        acolval = [4.0, 1.0]

        task.putacol(task.getnumvar() - 1, # column index
                     acolsub,
                     acolval)
```

After this operation the new problem is:

$$\begin{array}{lccccccccc} \text{maximize} & 1.5x_0 & + & 2.5x_1 & + & 3.0x_2 & + & 1.0x_3 & & \\ \text{subject to} & 3x_0 & + & 4x_1 & + & 3x_2 & + & 4x_3 & \leq & 100000, \\ & 3x_0 & + & 2x_1 & + & 3x_2 & & & \leq & 50000, \\ & 2x_0 & + & 3x_1 & + & 2x_2 & + & 1x_3 & \leq & 60000, \end{array} \tag{6.46}$$

and

$$x_0, x_1, x_2, x_3 \geq 0.$$

6.11.4 Appending Constraints

Now suppose we want to add a new stage to the production process called *Quality control* for which 30000 minutes are available. The time requirement for this stage is shown below:

Product no.	Quality control (minutes)
0	1
1	2
2	1
3	1

This corresponds to adding the constraint

$$x_0 + 2x_1 + x_2 + x_3 \leq 30000$$

to the problem. This is done as follows.

Listing 6.22: Adding a new constraint.

```
        ############# Add a new constraint ########################
        task.appendcons(1)
        numcon+=1

        # Set bounds on new constraint
        task.putconbound(task.getnumcon() - 1,
                         mosek.boundkey.up, -inf, 30000)

        # Put new values in the A matrix
        arowsub = [0, 1, 2, 3]
```

(continues on next page)

(continued from previous page)

```
        arowval = [1.0, 2.0, 1.0, 1.0]

        task.putarow(task.getnumcon() - 1, # row index
                     arowsub,
                     arowval)
```

Again, we can continue with re-optimizing the modified problem.

6.11.5 Changing bounds

One typical reoptimization scenario is to change bounds. Suppose for instance that we must operate with limited time resources, and we must change the upper bounds in the problem as follows:

Operation	Time available (before)	Time available (new)
Assembly	100000	80000
Polishing	50000	40000
Packing	60000	50000
Quality control	30000	22000

That means we would like to solve the problem:

$$\begin{array}{lrcrcrcrcl} \text{maximize} & 1.5x_0 & + & 2.5x_1 & + & 3.0x_2 & + & 1.0x_3 & & \\ \text{subject to} & 3x_0 & + & 4x_1 & + & 3x_2 & + & 4x_3 & \leq & 80000, \\ & 3x_0 & + & 2x_1 & + & 3x_2 & & & \leq & 40000, \\ & 2x_0 & + & 3x_1 & + & 2x_2 & + & 1x_3 & \leq & 50000, \\ & x_0 & + & 2x_1 & + & x_2 & + & x_3 & \leq & 22000. \end{array} \tag{6.47}$$

In this case all we need to do is redefine the upper bound vector for the constraints, as shown in the next listing.

Listing 6.23: Change constraint bounds.

```
        ############# Change constraint bounds ######################
        newbkc = [mosek.boundkey.up] * numcon
        newblc = [-inf] * numcon
        newbuc = [ 80000, 40000, 50000, 22000 ]

        task.putconboundslice(0, numcon, newbkc, newblc, newbuc)
```

Again, we can continue with re-optimizing the modified problem.

6.11.6 Advanced hot-start

If the optimizer used the data from the previous run to hot-start the optimizer for reoptimization, this will be indicated in the log:

```
Optimizer  - hotstart               : yes
```

When performing re-optimizations, instead of removing a basic variable it may be more efficient to fix the variable at zero and then remove it when the problem is re-optimized and it has left the basis. This makes it easier for **MOSEK** to restart the simplex optimizer.

6.12 Parallel optimization

In this section we demonstrate the method *Env.optimizebatch* which is a parallel optimization mechanism built-in in **MOSEK**. It has the following features:

- One license token checked out by the environment will be shared by the tasks.
- It allows to fine-tune the balance between the total number of threads in use by the parallel solver and the number of threads used for each individual task.
- It is very efficient for optimizing a large number of task of similar size, for example tasks obtained by cloning an initial task and changing some coefficients.

In the example below we simply load a few different tasks and optimize them together. When all tasks complete we access the response codes, solutions and other information in the standard way, as if each task was optimized separately.

Listing 6.24: Calling the parallel optimizer.

```
# Example of how to use env.optimizebatch()
# Optimizes tasks whose names were read from command line.
def main(argv):
  n = len(argv) - 1
  tasks = []

  threadpoolsize = 6                    # Size of thread pool available for all tasks

  with mosek.Env() as env:
    # Set up some example list of tasks to optimize
    for i in range(n):
      t = mosek.Task(env, 0, 0)
      t.readdata(argv[i+1])

      # We can set the number of threads for each task
      t.putintparam(mosek.iparam.num_threads, 2)
      tasks.append(t)

    # Optimize all the given tasks in parallel
    trm, res = env.optimizebatch(False,                 # No race
                                 -1.0,                  # No time limit
                                 threadpoolsize,
                                 tasks)                 # List of tasks to optimize

    for i in range(n):
      print("Task  {0}  res {1}   trm {2}   obj_val  {3}  time {4}".format(
             i,
             res[i],
             trm[i],
             tasks[i].getdouinf(mosek.dinfitem.intpnt_primal_obj),
             tasks[i].getdouinf(mosek.dinfitem.optimizer_time)))
```

Another, slightly more advanced application of the parallel optimizer is presented in Sec. 11.3.

6.13 Retrieving infeasibility certificates

When a continuous problem is declared as primal or dual infeasible, **MOSEK** provides a Farkas-type infeasibility certificate. If, as it happens in many cases, the problem is infeasible due to an unintended mistake in the formulation or because some individual constraint is too tight, then it is likely that infeasibility can be isolated to a few linear constraints/bounds that mutually contradict each other. In this case it is easy to identify the source of infeasibility. The tutorial in Sec. 8.3 has instructions on how to deal with this situation and debug it **by hand**. We recommend Sec. 8.3 as an introduction to infeasibility certificates and how to deal with infeasibilities in general.

Some users, however, would prefer to obtain the infeasibility certificate using Optimizer API for Python, for example in order to repair the issue automatically, display the information to the user, or perhaps simply because the infeasibility was one of the intended outcomes that should be analyzed in the code.

In this tutorial we show how to obtain such an infeasibility certificate with Optimizer API for Python in the most typical case, that is when the linear part of a problem is primal infeasible. A Farkas-type primal infeasibility certificate consists of the dual values of linear constraints and bounds. The names of duals corresponding to various parts of the problem are defined in Sec. 12.1.2. Each of the dual values (multipliers) indicates that a certain multiple of the corresponding constraint should be taken into account when forming the collection of mutually contradictory equalities/inequalities.

6.13.1 Example PINFEAS

For the purpose of this tutorial we use the same example as in Sec. 8.3, that is the primal infeasible problem

$$
\begin{array}{lllrcrcrcrcrcrcll}
\text{minimize} & & & x_0 & + & 2x_1 & + & 5x_2 & + & 2x_3 & + & x_4 & + & 2x_5 & + & x_6 & & \\
\text{subject to} & s_0: & & x_0 & + & x_1 & & & & & & & & & & & \leq & 200, \\
& s_1: & & & & & & x_2 & + & x_3 & & & & & & & \leq & 1000, \\
& s_2: & & & & & & & & & & x_4 & + & x_5 & + & x_6 & \leq & 1000, \\
& d_0: & & x_0 & & & & & & & + & x_4 & & & & & = & 1100, \\
& d_1: & & & & x_1 & & & & & & & & & & & = & 200, \\
& d_2: & & & & & & x_2 & + & & & & & x_5 & & & = & 500, \\
& d_3: & & & & & & & & x_3 & + & & & & & x_6 & = & 500, \\
& & & & & & & & & & & & & & & x_i & \geq & 0.
\end{array}
\tag{6.48}
$$

Checking infeasible status and adjusting settings

After the model has been solved we check that it is indeed infeasible. If yes, then we choose a threshold for when a certificate value is considered as an important contributor to infeasibility (ideally we would like to list all nonzero duals, but just like an optimal solution, an infeasibility certificate is also subject to floating-point rounding errors). All these steps are demonstrated in the snippet below:

```
    # Check problem status, we use the interior point solution
    if task.getprosta(soltype.itr) == prosta.prim_infeas:
        # Set the tolerance at which we consider a dual value as essential
        eps = 1e-7
```

Going through the certificate for a single item

We can define a fairly generic function which takes an array of lower and upper dual values and all other required data and prints out the positions of those entries whose dual values exceed the given threshold. These are precisely the values we are interested in:

```
# Analyzes and prints infeasibility contributing elements
# sl - dual values for lower bounds
# su - dual values for upper bounds
# eps - tolerance for when a nunzero dual value is significant
def analyzeCertificate(sl, su, eps):
```

(continues on next page)

(continued from previous page)

```
    n = len(sl)
    for i in range(n):
        if abs(sl[i]) > eps:
            print(f"#{i}: lower, dual = {sl[i]}")
        if abs(su[i]) > eps:
            print(f"#{i}: upper, dual = {su[i]}")
```

Full source code

All that remains is to call this function for all variable and constraint bounds for which we want to know their contribution to infeasibility. Putting all these pieces together we obtain the following full code:

Listing 6.25: Demonstrates how to retrieve a primal infeasibility certificate.

```
from mosek import *
import sys

# Set up a simple linear problem from the manual for test purposes
def testProblem():
    inf = 0.0
    task = Task()
    task.appendvars(7)
    task.appendcons(7)
    task.putclist(range(0, 7), [1, 2, 5, 2, 1, 2, 1])
    task.putaijlist([0,0,1,1,2,2,2,3,3,4,5,5,6,6],
                    [0,1,2,3,4,5,6,0,4,1,2,5,3,6],
                    [1] * 14)
    task.putconboundslice(0, 7, [boundkey.up]*3+[boundkey.fx]*4,
                                [-inf, -inf, -inf, 1100, 200, 500, 500],
                                [200, 1000, 1000, 1100, 200, 500, 500])
    task.putvarboundsliceconst(0, 7, boundkey.lo, 0, +inf)
    return task

# Analyzes and prints infeasibility contributing elements
# sl - dual values for lower bounds
# su - dual values for upper bounds
# eps - tolerance for when a nunzero dual value is significant
def analyzeCertificate(sl, su, eps):
    n = len(sl)
    for i in range(n):
        if abs(sl[i]) > eps:
            print(f"#{i}: lower, dual = {sl[i]}")
        if abs(su[i]) > eps:
            print(f"#{i}: upper, dual = {su[i]}")

def pinfeas():
    # In this example we set up a simple problem
    # One could use any task or a task read from a file
    task = testProblem()

    # Useful for debugging
    task.writedata("pinfeas.ptf")
    task.set_Stream(streamtype.log, sys.stdout.write)

    # Perform the optimization.
```

(continues on next page)

(continued from previous page)

```
    task.optimize()
    task.solutionsummary(streamtype.log)

    # Check problem status, we use the interior point solution
    if task.getprosta(soltype.itr) == prosta.prim_infeas:
        # Set the tolerance at which we consider a dual value as essential
        eps = 1e-7

        print("Variable bounds important for infeasibility: ")
        analyzeCertificate(task.getslx(soltype.itr), task.getsux(soltype.itr), eps)

        print("Constraint bounds important for infeasibility: ")
        analyzeCertificate(task.getslc(soltype.itr), task.getsuc(soltype.itr), eps)
    else:
        print("The problem is not primal infeasible, no certificate to show");

pinfeas()
```

Running this code will produce the following output:

```
Variable bounds important for infeasibility:
#6: lower, dual = 1.000000e+00
#7: lower, dual = 1.000000e+00
Constraint bounds important for infeasibility:
#1: upper, dual = 1.000000e+00
#3: upper, dual = 1.000000e+00
#4: lower, dual = 1.000000e+00
#5: lower, dual = 1.000000e+00
```

indicating the positions of bounds which appear in the infeasibility certificate with nonzero values. For a more in-depth treatment see the following sections:

- Sec. 11 for more advanced and complicated optimization examples.
- Sec. 11.1 for examples related to portfolio optimization.
- Sec. 12 for formal mathematical formulations of problems **MOSEK** can solve, dual problems and infeasibility certificates.

Chapter 7

Solver Interaction Tutorials

In this section we cover the interaction with the solver.

7.1 Environment and task

All interaction with Optimizer API for Python proceeds through one of two entry points: the **MOSEK tasks** and, to a lesser degree the **MOSEK environment** .

7.1.1 Task

The **MOSEK** task *Task* provides a representation of one optimization problem. It is the main interface through which all optimization is performed. Many tasks can be created and disposed of in one process.

A typical scenario for working with a task is shown below:

```
with mosek.Task() as task:
  # Define and solve an optimization problem here
  # ...
```

If a task is created outside of a context that ensures automatic garbage collection then it can be disposed of manually using *Task.__del__* .

7.1.2 Environment

The **MOSEK** environment *Env* coordinates access to **MOSEK** from the current process. It provides various general functionalities, in particular those related to license management, linear algebra, parallel optimization and certain other auxiliary functions. All tasks are explicitly or implicitly attached to some environment. It is recommended to have at most one environment per process.

Creating an environment is optional and only recommended for those users who will require some of the features it provides. Most users will **NOT need their own environment** and can skip this object. In this case **MOSEK** will internally create a global environment transparently for the user. This environment will not be accessible for the user.

A typical scenario for working with **MOSEK** through an explicit environment is shown below:

```
with mosek.Env() as env:
  # Create one or more tasks
  with env.Task() as task:
    # Define and solve an optimization problem here
    # ...
```

If an environment is created outside of a context that ensures automatic garbage collection then it can be disposed of manually using *Env.__del__* .

7.2 Accessing the solution

This section contains important information about the status of the solver and the status of the solution, which must be checked in order to properly interpret the results of the optimization.

7.2.1 Solver termination

The optimizer provides two status codes relevant for error handling:

- **Response code** of type *rescode*. It indicates if any unexpected error (such as an out of memory error, licensing error etc.) has occurred. The expected value for a successful optimization is *rescode.ok*.
- **Termination code**: It provides information about why the optimizer terminated, for instance if a predefined time limit has been reached. These are not errors, but ordinary events that can be expected (depending on parameter settings and the type of optimizer used).

If the optimization was successful then the method *Task.optimize* returns normally and its output is the termination code. If an error occurs then the method throws an exception, which contains the response code. See Sec. 7.3 for how to access it.

If a runtime error causes the program to crash during optimization, the first debugging step is to enable logging and check the log output. See Sec. 7.4.

If the optimization completes successfully, the next step is to check the solution status, as explained below.

7.2.2 Available solutions

MOSEK uses three kinds of optimizers and provides three types of solutions:

- **basic solution** from the simplex optimizer,
- **interior-point solution** from the interior-point optimizer,
- **integer solution** from the mixed-integer optimizer.

Under standard parameters settings the following solutions will be available for various problem types:

Table 7.1: Types of solutions available from **MOSEK**

	Simplex optimizer	Interior-point optimizer	Mixed-integer optimizer
Linear problem	*soltype.bas*	*soltype.itr*	
Nonlinear continuous problem		*soltype.itr*	
Problem with integer variables			*soltype.itg*

For linear problems the user can force a specific optimizer choice making only one of the two solutions available. For example, if the user disables basis identification, then only the interior point solution will be available for a linear problem. Numerical issues may cause one of the solutions to be unknown even if another one is feasible.

Not all components of a solution are always available. For example, there is no dual solution for integer problems and no dual conic variables from the simplex optimizer.

The user will always need to specify which solution should be accessed.

7.2.3 Problem and solution status

Assuming that the optimization terminated without errors, the next important step is to check the problem and solution status and availability of solutions. There is one for every type of solution, as explained above.

Problem status

Problem status (*prosta*) determines whether the problem is certified as feasible. Its values can roughly be divided into the following broad categories:

- **feasible** — the problem is feasible. For continuous problems and when the solver is run with default parameters, the feasibility status should ideally be *prosta.prim_and_dual_feas*.
- **primal/dual infeasible** — the problem is infeasible or unbounded or a combination of those. The exact problem status will indicate the type of infeasibility.
- **unknown** — the solver was unable to reach a conclusion, most likely due to numerical issues.

Solution status

Solution status (*solsta*) provides the information about what the solution values actually contain. The most important broad categories of values are:

- **optimal** (*solsta.optimal*) — the solution values are feasible and optimal.
- **certificate** — the solution is in fact a certificate of infeasibility (primal or dual, depending on the solution).
- **unknown/undefined** — the solver could not solve the problem or this type of solution is not available for a given problem.

Problem and solution status for each solution can be retrieved with *Task.getprosta* and *Task.getsolsta*, respectively.

The solution status determines the action to be taken. For example, in some cases a suboptimal solution may still be valuable and deserve attention. It is the user's responsibility to check the status and quality of the solution.

Typical status reports

Here are the most typical optimization outcomes described in terms of the problem and solution statuses. Note that these do not cover all possible situations that can occur.

Table 7.2: Continuous problems (solution status for interior-point and basic solution)

Outcome	Problem status	Solution status
Optimal	*prosta.prim_and_dual_feas*	*solsta.optimal*
Primal infeasible	*prosta.prim_infeas*	*solsta.prim_infeas_cer*
Dual infeasible (unbounded)	*prosta.dual_infeas*	*solsta.dual_infeas_cer*
Uncertain (stall, numerical issues, etc.)	*prosta.unknown*	*solsta.unknown*

Table 7.3: Integer problems (solution status for integer solution, others undefined)

Outcome	Problem status	Solution status
Integer optimal	*prosta.prim_feas*	*solsta.integer_optimal*
Infeasible	*prosta.prim_infeas*	*solsta.unknown*
Integer feasible point	*prosta.prim_feas*	*solsta.prim_feas*
No conclusion	*prosta.unknown*	*solsta.unknown*

7.2.4 Retrieving solution values

After the meaning and quality of the solution (or certificate) have been established, we can query for the actual numerical values. They can be accessed using:

- *Task.getprimalobj*, *Task.getdualobj* — the primal and dual objective value.
- *Task.getxx* — solution values for the variables.
- *Task.getsolution* — a full solution with primal and dual values

and many more specialized methods, see the *API reference*.

7.2.5 Source code example

Below is a source code example with a simple framework for assessing and retrieving the solution to a conic optimization problem.

Listing 7.1: Sample framework for checking optimization result.

```
import mosek
import sys

# A log message
def streamprinter(text):
    sys.stdout.write(text)
    sys.stdout.flush()

def main(args):
  filename = args[0] if len(args) >= 1 else "../data/cqo1.mps"

  try:
    with mosek.Task() as task:
      # (Optional) set a log stream
      # task.set_Stream(mosek.streamtype.log, streamprinter)

      # (Optional) uncomment to see what happens when solution status is unknown
      # task.putintparam(mosek.iparam.intpnt_max_iterations, 1)

      # In this example we read data from a file
      task.readdata(filename)

      # Optimize
      trmcode = task.optimize()
      task.solutionsummary(mosek.streamtype.log)

      # We expect solution status OPTIMAL
      solsta = task.getsolsta(mosek.soltype.itr)

      if solsta == mosek.solsta.optimal:
        # Optimal solution. Fetch and print it.
        print("An optimal interior-point solution is located.")
        numvar = task.getnumvar()
        xx = task.getxx(mosek.soltype.itr)
        for i in range(numvar):
          print("x[{0}] = {1}".format(i, xx[i]))

      elif solsta == mosek.solsta.dual_infeas_cer:
        print("Dual infeasibility certificate found.")
```

(continues on next page)

(continued from previous page)

```
      elif solsta == mosek.solsta.prim_infeas_cer:
        print("Primal infeasibility certificate found.")

      elif solsta == mosek.solsta.unknown:
        # The solutions status is unknown. The termination code
        # indicates why the optimizer terminated prematurely.
        print("The solution status is unknown.")
        symname, desc = mosek.Env.getcodedesc(trmcode)
        print("   Termination code: {0} {1}".format(symname, desc))

      else:
        print("An unexpected solution status {0} is obtained.".format(str(solsta)))

  except mosek.Error as e:
      print("Unexpected error ({0}) {1}".format(e.errno, e.msg))

if __name__ == '__main__':
    main(sys.argv[1:])
```

7.3 Errors and exceptions

Exceptions

Almost every function in Optimizer API for Python can throw an exception informing that the requested operation was not performed correctly, and indicating the type of error that occurred. This is the case in situations such as for instance:

- referencing a nonexisting variable (for example with too large index),
- defining an invalid value for a parameter,
- accessing an undefined solution,
- repeating a variable name, etc.

It is therefore a good idea to catch exceptions of type *Error*. The one case where it is *extremely important* to do so is when *Task.optimize* is invoked. We will say more about this in Sec. 7.2.

The exception contains a *response code* (element of the enum *rescode*) and short diagnostic messages. They can be accessed as in the following example.

```
try:
  task.putdouparam(mosek.dparam.intpnt_co_tol_rel_gap, -1.0e-7)
except mosek.Error as e:
  print("Response code {0}\nMessage       {1}".format(e.errno, e.msg))
```

It will produce as output:

```
Response code rescode.err_param_is_too_small
Message       The parameter value -1e-07 is too small for parameter 'MSK_DPAR_INTPNT_
↪CO_TOL_REL_GAP'.
```

Another way to obtain a human-readable string corresponding to a response code is the method *Env.getcodedesc*. A full list of exceptions, as well as response codes, can be found in the *API reference*.

Optimizer errors and warnings

The optimizer may also produce warning messages. They indicate non-critical but important events, that will not prevent solver execution, but may be an indication that something in the optimization problem might be improved. Warning messages are normally printed to a log stream (see Sec. 7.4). A typical warning is, for example:

```
MOSEK warning 53: A numerically large upper bound value  6.6e+09 is specified for␣
↪constraint 'C69200' (46020).
```

Warnings can also be suppressed by setting the *iparam.max_num_warnings* parameter to zero, if they are well-understood.

Error and solution status handling example

Below is a source code example with a simple framework for handling major errors when assessing and retrieving the solution to a conic optimization problem.

Listing 7.2: Sample framework for checking optimization result.

```
import mosek
import sys

# A log message
def streamprinter(text):
    sys.stdout.write(text)
    sys.stdout.flush()

def main(args):
  filename = args[0] if len(args) >= 1 else "../data/cqo1.mps"

  try:
    with mosek.Task() as task:
      # (Optional) set a log stream
      # task.set_Stream(mosek.streamtype.log, streamprinter)

      # (Optional) uncomment to see what happens when solution status is unknown
      # task.putintparam(mosek.iparam.intpnt_max_iterations, 1)

      # In this example we read data from a file
      task.readdata(filename)

      # Optimize
      trmcode = task.optimize()
      task.solutionsummary(mosek.streamtype.log)

      # We expect solution status OPTIMAL
      solsta = task.getsolsta(mosek.soltype.itr)

      if solsta == mosek.solsta.optimal:
        # Optimal solution. Fetch and print it.
        print("An optimal interior-point solution is located.")
        numvar = task.getnumvar()
        xx = task.getxx(mosek.soltype.itr)
        for i in range(numvar):
          print("x[{0}] = {1}".format(i, xx[i]))

      elif solsta == mosek.solsta.dual_infeas_cer:
        print("Dual infeasibility certificate found.")
```

(continues on next page)

(continued from previous page)

```
      elif solsta == mosek.solsta.prim_infeas_cer:
        print("Primal infeasibility certificate found.")

      elif solsta == mosek.solsta.unknown:
        # The solutions status is unknown. The termination code
        # indicates why the optimizer terminated prematurely.
        print("The solution status is unknown.")
        symname, desc = mosek.Env.getcodedesc(trmcode)
        print("   Termination code: {0} {1}".format(symname, desc))

      else:
        print("An unexpected solution status {0} is obtained.".format(str(solsta)))

  except mosek.Error as e:
      print("Unexpected error ({0}) {1}".format(e.errno, e.msg))

if __name__ == '__main__':
    main(sys.argv[1:])
```

7.4 Input/Output

The logging and I/O features are provided mainly by the **MOSEK** task and to some extent by the **MOSEK** environment objects.

7.4.1 Stream logging

By default the solver runs silently and does not produce any output to the console or otherwise. However, the log output can be redirected to a user-defined output stream or stream callback function. The log output is analogous to the one produced by the command-line version of **MOSEK**.

The log messages are partitioned in three streams:

- messages, *streamtype.msg*
- warnings, *streamtype.wrn*
- errors, *streamtype.err*

These streams are aggregated in the *streamtype.log* stream. A stream handler can be defined for each stream separately.

A stream handler is simply a user-defined function of type *streamfunc* that accepts a string, for example:

```
def myStream(msg):
    sys.stdout.write(msg)
    sys.stdout.flush()
```

It is attached to a stream as follows:

```
task.set_Stream(streamtype.log,myStream)
```

The stream can be detached by calling

```
task.set_Stream(streamtype.log,None)
```

A log stream can also be redirected to a file:

```
task.linkfiletostream(mosek.streamtype.log, "mosek.log", 0);
```

After optimization is completed an additional short summary of the solution and optimization process can be printed to any stream using the method *Task.solutionsummary*.

7.4.2 Log verbosity

The logging verbosity can be controlled by setting the relevant parameters, as for instance

- *iparam.log*,
- *iparam.log_intpnt*,
- *iparam.log_mio*,
- *iparam.log_cut_second_opt*,
- *iparam.log_sim*.

Each parameter controls the output level of a specific functionality or algorithm. The main switch is *iparam.log* which affect the whole output. The actual log level for a specific functionality is determined as the minimum between *iparam.log* and the relevant parameter. For instance, the log level for the output produce by the interior-point algorithm is tuned by the *iparam.log_intpnt*; the actual log level is defined by the minimum between *iparam.log* and *iparam.log_intpnt*.

Tuning the solver verbosity may require adjusting several parameters. It must be noticed that verbose logging is supposed to be of interest during debugging and tuning. When output is no more of interest, the user can easily disable it globally with *iparam.log*. Larger values of *iparam.log* do not necessarily result in increased output.

By default **MOSEK** will reduce the amount of log information after the first optimization on a given problem. To get full log output on subsequent re-optimizations set *iparam.log_cut_second_opt* to zero.

7.4.3 Saving a problem to a file

An optimization problem can be dumped to a file using the method *Task.writedata*. The file format will be determined from the extension of the filename. Supported formats are listed in Sec. 16 together with a table of problem types supported by each.

For instance the problem can be written to a human-readable PTF file (see Sec. 16.5) with

```
task.writedata("data.ptf")
```

All formats can be compressed with `gzip` by appending the `.gz` extension, and with `ZStandard` by appending the `.zst` extension, for example

```
task.writedata("data.task.gz")
```

Some remarks:

- Unnamed variables are given generic names. It is therefore recommended to use meaningful variable names if the problem file is meant to be human-readable.
- The `task` format is **MOSEK**'s native file format which contains all the problem data as well as solver settings.

7.4.4 Reading a problem from a file

A problem saved in any of the supported file formats can be read directly into a task using *Task.readdata*. The task must be created in advance. Afterwards the problem can be optimized, modified, etc. If the file contained solutions, then are also imported, but the status of any solution will be set to *solsta.unknown* (solutions can also be read separately using *Task.readsolution*). If the file contains parameters, they will be set accordingly.

```
task = env.Task()
try:
    task.readdata("file.task.gz")
    task.optimize()
except mosek.Error:
    print("Problem reading the file")
```

7.5 Setting solver parameters

MOSEK comes with a large number of parameters that allows the user to tune the behavior of the optimizer. The typical settings which can be changed with solver parameters include:

- choice of the optimizer for linear problems,
- choice of primal/dual solver,
- turning presolve on/off,
- turning heuristics in the mixed-integer optimizer on/off,
- level of multi-threading,
- feasibility tolerances,
- solver termination criteria,
- behaviour of the license manager,

and more. All parameters have default settings which will be suitable for most typical users.
The API reference contains:

- *Full list of parameters*
- *List of parameters grouped by topic*

Setting parameters

Each parameter is identified by a unique name. There are three types of parameters depending on the values they take:

- Integer parameters. They take either either simple integer values or values from an enumeration provided for readability and compatibility of the code. Set with *Task.putintparam*.
- Double (floating point) parameters. Set with *Task.putdouparam*.
- String parameters. Set with *Task.putstrparam*.

There are also parameter setting functions which operate fully on symbolic strings containing generic command-line style names of parameters and their values. See the example below. The optimizer will try to convert the given argument to the exact expected type, and will error if that fails.
If an incorrect value is provided then the parameter is left unchanged.
For example, the following piece of code sets up some parameters before solving a problem.

Listing 7.3: Parameter setting example.

```
# Set log level (integer parameter)
task.putintparam(mosek.iparam.log, 1)
# Select interior-point optimizer... (integer parameter)
task.putintparam(mosek.iparam.optimizer, mosek.optimizertype.intpnt)
# ... without basis identification (integer parameter)
task.putintparam(mosek.iparam.intpnt_basis, mosek.basindtype.never)
# Set relative gap tolerance (double parameter)
task.putdouparam(mosek.dparam.intpnt_co_tol_rel_gap, 1.0e-7)

# The same using explicit string names
task.putparam     ("MSK_DPAR_INTPNT_CO_TOL_REL_GAP", "1.0e-7")
task.putnadouparam("MSK_DPAR_INTPNT_CO_TOL_REL_GAP",  1.0e-7 )

# Incorrect value
try:
    task.putdouparam(mosek.dparam.intpnt_co_tol_rel_gap, -1.0)
except:
    print('Wrong parameter value')
```

Reading parameter values

The functions *Task.getintparam*, *Task.getdouparam*, *Task.getstrparam* can be used to inspect the current value of a parameter, for example:

```
param = task.getdouparam(mosek.dparam.intpnt_co_tol_rel_gap)
print('Current value for parameter intpnt_co_tol_rel_gap = {}'.format(param))
```

7.6 Retrieving information items

After the optimization the user has access to the solution as well as to a report containing a large amount of additional *information items*. For example, one can obtain information about:

- **timing**: total optimization time, time spent in various optimizer subroutines, number of iterations, etc.
- **solution quality**: feasibility measures, solution norms, constraint and bound violations, etc.
- **problem structure**: counts of variables of different types, constraints, nonzeros, etc.
- **integer optimizer**: integrality gap, objective bound, number of cuts, etc.

and more. Information items are numerical values of integer, long integer or double type. The full list can be found in the API reference:

- *Double*
- *Integer*
- *Long*

Certain information items make sense, and are made available, also *during* the optimization process. They can be accessed from a callback function, see Sec. 7.7 for details.

Remark

For efficiency reasons, not all information items are automatically computed after optimization. To force all information items to be updated use the parameter *iparam.auto_update_sol_info*.

Retrieving the values

Values of information items are fetched using one of the methods

- *Task.getdouinf* for a double information item,
- *Task.getintinf* for an integer information item,
- *Task.getlintinf* for a long integer information item.

Each information item is identified by a unique name. The example below reads two pieces of data from the solver: total optimization time and the number of interior-point iterations.

Listing 7.4: Information items example.

```
tm = task.getdouinf(mosek.dinfitem.optimizer_time)
it = task.getintinf(mosek.iinfitem.intpnt_iter)

print('Time: {0}\nIterations: {1}'.format(tm,it))
```

7.7 Progress and data callback

Callbacks are a very useful mechanism that allow the caller to track the progress of the **MOSEK** optimizer. A callback function provided by the user is regularly called during the optimization and can be used to

- obtain a customized log of the solver execution,
- collect information for debugging purposes or
- ask the solver to terminate.

Optimizer API for Python has the following callback mechanisms:

- **progress callback**, which provides only the basic status of the solver.
- **data callback**, which provides the solver status and a complete set of information items that describe the progress of the optimizer in detail.

Warning

The callbacks functions *must not* invoke any functions of the solver, environment or task. Otherwise the state of the solver and its outcome are undefined. The only exception is the possibility to retrieve an integer solution, see below.

Retrieving mixed-integer solutions

If the mixed-integer optimizer is used, the callback will take place, in particular, every time an improved integer solution is found. In that case it is possible to retrieve the current values of the best integer solution from within the callback function. It can be useful for implementing complex termination criteria for integer optimization. The example in Listing 7.5 shows how to do it by handling the callback code `callbackcode.new_int_mio`.

7.7.1 Data callback

In the data callback **MOSEK** passes a callback code and values of all information items to a user-defined function. The callback function is called, in particular, at the beginning of each iteration of the interior-point optimizer. For the simplex optimizers `iparam.log_sim_freq` controls how frequently the call-back is called.

The callback is set by calling the method `Task.set_InfoCallback` and providing a handle to a user-defined function `callbackfunc`.

Non-zero return value of the callback function indicates that the optimizer should be terminated.

7.7.2 Progress callback

In the progress callback **MOSEK** provides a single code indicating the current stage of the optimization process.

The callback is set by calling the method `Task.set_Progress` and providing a handle to a user-defined function `progresscallbackfunc`.

Non-zero return value of the callback function indicates that the optimizer should be terminated.

7.7.3 Working example: Data callback

The following example defines a data callback function that prints out some of the information items. It interrupts the solver after a certain time limit.

Listing 7.5: An example of a data callback function.

```
def makeUserCallback(maxtime, task):
    pass

    def userCallback(caller,
                     douinf,
                     intinf,
                     lintinf):
        opttime = 0.0

        if caller == callbackcode.begin_intpnt:
            print("Starting interior-point optimizer")
        elif caller == callbackcode.intpnt:
            itrn = intinf[iinfitem.intpnt_iter]
            pobj = douinf[dinfitem.intpnt_primal_obj]
            dobj = douinf[dinfitem.intpnt_dual_obj]
            stime = douinf[dinfitem.intpnt_time]
            opttime = douinf[dinfitem.optimizer_time]

            print("Iterations: %-3d" % itrn)
            print("  Elapsed time: %6.2f(%.2f) " % (opttime, stime))
            print("  Primal obj.: %-18.6e  Dual obj.: %-18.6e" % (pobj, dobj))
        elif caller == callbackcode.end_intpnt:
            print("Interior-point optimizer finished.")
        elif caller == callbackcode.begin_primal_simplex:
            print("Primal simplex optimizer started.")
        elif caller == callbackcode.update_primal_simplex:
            itrn = intinf[iinfitem.sim_primal_iter]
            pobj = douinf[dinfitem.sim_obj]
            stime = douinf[dinfitem.sim_time]
            opttime = douinf[dinfitem.optimizer_time]

            print("Iterations: %-3d" % itrn)
            print("  Elapsed time: %6.2f(%.2f)" % (opttime, stime))
            print("  Obj.: %-18.6e" % pobj)
        elif caller == callbackcode.end_primal_simplex:
            print("Primal simplex optimizer finished.")
        elif caller == callbackcode.begin_dual_simplex:
            print("Dual simplex optimizer started.")
        elif caller == callbackcode.update_dual_simplex:
            itrn = intinf[iinfitem.sim_dual_iter]
            pobj = douinf[dinfitem.sim_obj]
            stime = douinf[dinfitem.sim_time]
            opttime = douinf[dinfitem.optimizer_time]
            print("Iterations: %-3d" % itrn)
            print("  Elapsed time: %6.2f(%.2f)" % (opttime, stime))
            print("  Obj.: %-18.6e" % pobj)
        elif caller == callbackcode.end_dual_simplex:
            print("Dual simplex optimizer finished.")
        elif caller == callbackcode.new_int_mio:
            print("New integer solution has been located.")
            xx = task.getxx(soltype.itg)
            print(xx)
            print("Obj.: %f" % douinf[dinfitem.mio_obj_int])
        else:
            pass
```

(continues on next page)

(continued from previous page)

```
        if opttime >= maxtime:
            # mosek is spending too much time. Terminate it.
            print("Terminating.")
            return 1

        return 0
    return userCallback
```

Assuming that we have defined a task `task` and a time limit `maxtime`, the callback function is attached as follows:

Listing 7.6: Attaching the data callback function to the model.

```
        usercallback = makeUserCallback(maxtime=0.05, task=task)
        task.set_InfoCallback(usercallback)
```

7.8 MOSEK OptServer

MOSEK provides an easy way to offload optimization problem to a remote server. This section demonstrates related functionalities from the client side, i.e. sending optimization tasks to the remote server and retrieving solutions.

Setting up and configuring the remote server is described in a separate manual for the OptServer.

The `URL` of the remote server required in all client-side calls should be a string of the form `http://host:port` or `https://host:port`.

7.8.1 Synchronous Remote Optimization

In synchronous mode the client sends an optimization problem to the server and blocks, waiting for the optimization to end. Once the result has been received, the program can continue. This is the simplest mode all it takes is to provide the address of the server before starting optimization. The rest of the code remains untouched.

Note that it is impossible to recover the job in case of a broken connection.

Source code example

Listing 7.7: Using the OptServer in synchronous mode.

```
import mosek
import sys

def streamprinter(msg):
    sys.stdout.write(msg)
    sys.stdout.flush()

if len(sys.argv) <= 2:
    print("Missing argument, syntax is:")
    print("  opt_server_sync inputfile addr [certpath]")
else:

    inputfile = sys.argv[1]
    serveraddr = sys.argv[2]
    tlscert = None if len(sys.argv) < 4 else sys.argv[3]

    with mosek.Task() as task:
```

(continues on next page)

(continued from previous page)

```
        task.set_Stream(mosek.streamtype.log, streamprinter)

        # We assume that a problem file was given as the first command
        # line argument (received in `argv')
        task.readdata(inputfile)

        # Set OptServer URL
        task.putoptserverhost(serveraddr)

        # Path to certificate, if any
        if tlscert is not None:
            task.putstrparam(mosek.sparam.remote_tls_cert_path, tlscert)

        # Solve the problem remotely, no access token
        trm = task.optimize()

        # Print a summary of the solution
        task.solutionsummary(mosek.streamtype.log)
```

7.8.2 Asynchronous Remote Optimization

In asynchronous mode the client sends a job to the remote server and the execution of the client code continues. In particular, it is the client's responsibility to periodically check the optimization status and, when ready, fetch the results. The client can also interrupt optimization. The most relevant methods are:

- *Task.asyncoptimize* : Offload the optimization task to a solver server.
- *Task.asyncpoll* : Request information about the status of the remote job.
- *Task.asyncgetresult* : Request the results from a completed remote job.
- *Task.asyncstop* : Terminate a remote job.

Source code example

In the example below the program enters in a polling loop that regularly checks whether the result of the optimization is available.

Listing 7.8: Using the OptServer in asynchronous mode.

```
import mosek
import sys
import time

def streamprinter(msg):
    sys.stdout.write(msg)
    sys.stdout.flush()

if len(sys.argv) < 4:
    print("Missing argument, syntax is:")
    print("  opt-server-async inputfile host:port numpolls [cert]")
else:
    filename = sys.argv[1]
    addr = sys.argv[2]
    numpolls = int(sys.argv[3])
    token = None
    cert = None if len(sys.argv) < 5 else sys.argv[4]
```

(continues on next page)

(continued from previous page)

```
with mosek.Task() as task:
    print("reading task from file")
    task.readdata(filename)

    if cert is not None:
        task.putstrparam(mosek.sparam.remote_tls_cert_path,cert)

    print("Solve the problem remotely (async)")
    token = task.asyncoptimize(addr,"")

print("Task token: %s" % token)

with mosek.Task() as task:
    task.readdata(filename)

    if cert is not None:
        task.putstrparam(mosek.sparam.remote_tls_cert_path,cert)
    task.set_Stream(mosek.streamtype.log, streamprinter)

    i = 0

    while i < numpolls:

        time.sleep(0.1)

        print("poll %d..." % i)
        respavailable, res, trm = task.asyncpoll(addr,
                                                 "",
                                                 token)

        print("done!")

        if respavailable:
            print("solution available!")

            respavailable, res, trm = task.asyncgetresult(addr,
                                                          "",
                                                          token)

            task.solutionsummary(mosek.streamtype.log)
            break

        i = i + 1

        if i == numpolls:
            print("max number of polls reached, stopping host.")
            task.asyncstop(addr,"", token)
```

Chapter 8

Debugging Tutorials

This collection of tutorials contains basic techniques for debugging optimization problems using tools available in **MOSEK**: optimizer log, solution summary, infeasibility report, command-line tools. It is intended as a first line of technical help for issues such as: Why do I get solution status *unknown* and how can I fix it? Why is my model infeasible while it shouldn't be? Should I change some parameters? Can the model solve faster? etc.

The major steps when debugging a model are always:

- Enable log output. See Sec. 7.4.1 for how to do it. In the simplest case:

  Attach a log handler function to the log stream:

  ```
  task.set_Stream(streamtype.log, sys.stdout.write)
  ```

  and include solution summary after the optimization:

  ```
  task.optimize()
  task.solutionsummary(streamtype.log)
  ```

- Run the optimization and analyze the log output, see Sec. 8.1. In particular:
  - check if the problem setup (number of constraints/variables etc.) matches your expectation.
  - check solution summary and solution status.
- Dump the problem to disk if necessary to continue analysis. See Sec. 7.4.3.
  - use a human-readable text format, preferably `*.ptf` if you want to check the problem structure by hand. Assign names to variables and constraints to make them easier to identify.

    ```
    task.writedata("data.ptf")
    ```

  - use the **MOSEK** native format `*.task.gz` when submitting a bug report or support question.

    ```
    task.writedata("data.task.gz")
    ```

- Fix problem setup, improve the model, locate infeasibility or adjust parameters, depending on the diagnosis.

See the following sections for details.

8.1 Understanding optimizer log

The optimizer produces a log which splits roughly into four sections:

1. summary of the input data,
2. presolve and other pre-optimize problem setup stages,
3. actual optimizer iterations,
4. solution summary.

In this tutorial we show how to analyze the most important parts of the log when initially debugging a model: input data (1) and solution summary (4). For the iterations log (3) see Sec. 13.3.4 or Sec. 13.4.3.

8.1.1 Input data

If **MOSEK** behaves very far from expectations it may be due to errors in problem setup. The log file will begin with a summary of the structure of the problem, which looks for instance like:

```
Problem
  Name                   :
  Objective sense        : minimize
  Type                   : CONIC (conic optimization problem)
  Constraints            : 234
  Affine conic cons.     : 5348 (6444 rows)
  Disjunctive cons.      : 0
  Cones                  : 0
  Scalar variables       : 20693
  Matrix variables       : 1 (scalarized: 45)
  Integer variables      : 0
```

This can be consulted to eliminate simple errors: wrong objective sense, wrong number of variables etc. Note that some modeling tools can introduce additional variables and constraints to the model and perturb the model even further (such as by dualizing). In most **MOSEK** APIs the problem dimensions should match exactly what the user specified.

If this is not sufficient a bit more information can be obtained by dumping the problem to a file (see Sec. 8) and using the `anapro` option of any of the command line tools. It can also be done directly with the function *Task.analyzeproblem*. This will produce a longer summary similar to:

```
** Variables
scalar: 20414     integer: 0        matrix: 0
low: 2082         up: 5014          ranged: 0         free: 12892       fixed: 426

** Constraints
all: 20413
low: 10028        up: 0             ranged: 0         free: 0           fixed: 10385

** Affine conic constraints (ACC)
QUAD: 1           dims: 2865: 1
RQUAD: 2507       dims: 3: 2507

** Problem data (numerics)
|c|               nnz: 10028        min=2.09e-05      max=1.00e+00
|A|               nnz: 597023       min=1.17e-10      max=1.00e+00
blx               fin: 2508         min=-3.60e+09     max=2.75e+05
bux               fin: 5440         min=0.00e+00      max=2.94e+08
blc               fin: 20413        min=-7.61e+05     max=7.61e+05
buc               fin: 10385        min=-5.00e-01     max=0.00e+00
```

(continues on next page)

(continued from previous page)

```
|F|               nnz: 612301       min=8.29e-06      max=9.31e+01
|g|               nnz: 1203         min=5.00e-03      max=1.00e+00
```

Again, this can be used to detect simple errors, such as:

- Wrong type of conic constraint was used or it has wrong dimension.
- The bounds for variables or constraints are incorrect or incomplete. Check if you defined bound keys for all variables. A variable for which no bound was defined is by default fixed at 0.
- The model is otherwise incomplete.
- Suspicious values of coefficients.
- For various data sizes the model does not scale as expected.

Finally saving the problem in a human-friendly text format such as LP or PTF (see Sec. 8) and analyzing it by hand can reveal if the model is correct.

Warnings and errors

At this stage the user can encounter warnings which should not be ignored, unless they are well-understood. They can also serve as hints as to numerical issues with the problem data. A typical warning of this kind is

```
MOSEK warning 53: A numerically large upper bound value  2.9e+08 is specified for␣
↪variable 'absh[107]' (2613).
```

Warnings do not stop the problem setup. If, on the other hand, an error occurs then the model will become invalid. The user should make sure to test for errors/exceptions from all API calls that set up the problem and validate the data. See Sec. 7.3 for more details.

8.1.2 Solution summary

The last item in the log is the solution summary. In the Optimizer API it is only printed by invoking the function *Task.solutionsummary*.

Continuous problem

Optimal solution

A typical solution summary for a continuous (linear, conic, quadratic) problem looks like:

```
Problem status  : PRIMAL_AND_DUAL_FEASIBLE
Solution status : OPTIMAL
Primal.  obj: 8.7560516107e+01    nrm: 1e+02    Viol.  con: 3e-12    var: 0e+00    ␣
↪acc: 3e-11
Dual.    obj: 8.7560521345e+01    nrm: 1e+00    Viol.  con: 5e-09    var: 9e-11    ␣
↪acc: 0e+00
```

It contains the following elements:

- Problem and solution status. For details see Sec. 7.2.3.
- A summary of the primal solution: objective value, infinity norm of the solution vector and maximal violations of variables and constraints of different types. The violation of a linear constraint such as $a^T x \leq b$ is $\max(a^T x - b, 0)$. The violation of a conic constraint is the distance to the cone.
- The same for the dual solution.

The features of the solution summary which characterize a very good and accurate solution and a well-posed model are:

- **Status:** The solution status is `OPTIMAL`.

- **Duality gap:** The primal and dual objective values are (almost) identical, which proves the solution is (almost) optimal.
- **Norms:** Ideally the norms of the solution and the objective values should not be too large. This of course depends on the input data, but a huge solution norm can be an indicator of issues with the scaling, conditioning and/or well-posedness of the model. It may also indicate that the problem is borderline between feasibility and infeasibility and sensitive to small perturbations in this respect.
- **Violations:** The violations are close to zero, which proves the solution is (almost) feasible. Observe that due to rounding errors it can be expected that the violations are proportional to the norm (`nrm:`) of the solution. It is rarely the case that violations are exactly zero.

Solution status UNKNOWN

A typical example with solution status `UNKNOWN` due to numerical problems will look like:

```
Problem status  : UNKNOWN
Solution status : UNKNOWN
Primal.  obj: 1.3821656824e+01    nrm: 1e+01    Viol.  con: 2e-03    var: 0e+00    ␣
↪acc: 0e+00
Dual.    obj: 3.0119004098e-01    nrm: 5e+07    Viol.  con: 4e-16    var: 1e-01    ␣
↪acc: 0e+00
```

Note that:

- The primal and dual objective are very different.
- The dual solution has very large norm.
- There are considerable violations so the solution is likely far from feasible.

Follow the hints in Sec. 8.2 to resolve the issue.

Solution status UNKNOWN with a potentially useful solution

Solution status `UNKNOWN` does not necessarily mean that the solution is completely useless. It only means that the solver was unable to make any more progress due to numerical difficulties, and it was not able to reach the accuracy required by the termination criteria (see Sec. 13.3.2). Consider for instance:

```
Problem status  : UNKNOWN
Solution status : UNKNOWN
Primal.  obj: 3.4531019648e+04    nrm: 1e+05    Viol.  con: 7e-02    var: 0e+00    ␣
↪acc: 0e+00
Dual.    obj: 3.4529720645e+04    nrm: 8e+03    Viol.  con: 1e-04    var: 2e-04    ␣
↪acc: 0e+00
```

Such a solution may still be useful, and it is always up to the user to decide. It may be a good enough approximation of the optimal point. For example, the large constraint violation may be due to the fact that one constraint contained a huge coefficient.

Infeasibility certificate

A primal infeasibility certificate is stored in the dual variables:

```
Problem status  : PRIMAL_INFEASIBLE
Solution status : PRIMAL_INFEASIBLE_CER
Dual.    obj: 2.9238975853e+02    nrm: 6e+02    Viol.  con: 0e+00    var: 1e-11    ␣
↪acc: 0e+00
```

It is a Farkas-type certificate as described in Sec. 12.2.2. In particular, for a good certificate:

- The dual objective is positive for a minimization problem, negative for a maximization problem. Ideally it is well bounded away from zero.

- The norm is not too big and the violations are small (as for a solution).

If the model was not expected to be infeasible, the likely cause is an error in the problem formulation. Use the hints in Sec. 8.1.1 and Sec. 8.3 to locate the issue.

Just like a solution, the infeasibility certificate can be of better or worse quality. The infeasibility certificate above is very solid. However, there can be less clear-cut cases, such as for example:

```
Problem status  : PRIMAL_INFEASIBLE
Solution status : PRIMAL_INFEASIBLE_CER
Dual.    obj: 1.6378689238e-06    nrm: 6e+05    Viol.  con: 7e-03    var: 2e-04    ␣
↪acc: 0e+00
```

This infeasibility certificate is more dubious because the dual objective is positive, but barely so in comparison with the large violations. It also has rather large norm. This is more likely an indication that the problem is borderline between feasibility and infeasibility or simply ill-posed and sensitive to tiny variations in input data. See Sec. 8.3 and Sec. 8.2.

The same remarks apply to dual infeasibility (i.e. unboundedness) certificates. Here the primal objective should be negative a minimization problem and positive for a maximization problem.

8.1.3 Mixed-integer problem

Optimal integer solution

For a mixed-integer problem there is no dual solution and a typical optimal solution report will look as follows:

```
Problem status  : PRIMAL_FEASIBLE
Solution status : INTEGER_OPTIMAL
Primal.  obj: 6.0111122960e+06    nrm: 1e+03    Viol.  con: 2e-13    var: 2e-14    ␣
↪itg: 5e-15
```

The interpretation of all elements is as for a continuous problem. The additional field `itg` denotes the maximum violation of an integer variable from being an exact integer.

Feasible integer solution

If the solver found an integer solution but did not prove optimality, for instance because of a time limit, the solution status will be `PRIMAL_FEASIBLE`:

```
Problem status  : PRIMAL_FEASIBLE
Solution status : PRIMAL_FEASIBLE
Primal.  obj: 6.0114607792e+06    nrm: 1e+03    Viol.  con: 2e-13    var: 2e-13    ␣
↪itg: 4e-15
```

In this case it is valuable to go back to the optimizer summary to see how good the best solution is:

```
31       35       1        0        6.0114607792e+06     6.0078960892e+06     0.06    ␣
↪    4.1

Objective of best integer solution : 6.011460779193e+06
Best objective bound               : 6.007896089225e+06
```

In this case the best integer solution found has objective value `6.011460779193e+06`, the best proved lower bound is `6.007896089225e+06` and so the solution is guaranteed to be within 0.06% from optimum. The same data can be obtained as information items through an API. See also Sec. 13.4 for more details.

Infeasible problem

If the problem is declared infeasible the summary is simply

```
Problem status  : PRIMAL_INFEASIBLE
Solution status : UNKNOWN
Primal.  obj: 0.0000000000e+00    nrm: 0e+00    Viol.  con: 0e+00    var: 0e+00    ␣
↪itg: 0e+00
```

If infeasibility was not expected, consult Sec. 8.3.

8.2 Addressing numerical issues

The suggestions in this section should help diagnose and solve issues with numerical instability, in particular `UNKNOWN` solution status or solutions with large violations. Since numerically stable models tend to solve faster, following these hints can also dramatically shorten solution times.

We always recommend that issues of this kind are addressed by reformulating or rescaling the model, since it is the modeler who has the best insight into the structure of the problem and can fix the cause of the issue.

Some information about the numerical properties of the data can be obtained by dumping the problem to a file (see Sec. 8) and using the `anapro` option of any of the command line tools. It can also be done directly with the function *`Task.analyzeproblem`*.

8.2.1 Formulating problems

Scaling

Make sure that all the data in the problem are of comparable orders of magnitude. This applies especially to the linear constraint matrix. Use Sec. 8.1.1 if necessary. For example a report such as

```
|A|               nnz: 597023       min=1.17e-6      max=2.21e+5
```

means that the ratio of largest to smallest elements in `A` is 10^{11}. In this case the user should rescale or reformulate the model to avoid such spread which makes it difficult for **MOSEK** to scale the problem internally. In many cases it may be possible to change the units, i.e. express the model in terms of rescaled variables (for instance work with millions of dollars instead of dollars, etc.).

Similarly, if the objective contains very different coefficients, say

$$\text{maximize } 10^{10}x + y$$

then it is likely to lead to inaccuracies. The objective will be dominated by the contribution from x and y will become insignificant.

Removing huge bounds

Never use a very large number as replacement for ∞. Instead define the variable or constraint as unbounded from below/above. Similarly, avoid artificial huge bounds if you expect they will not become tight in the optimal solution.

Avoiding linear dependencies

As much as possible try to avoid linear dependencies and near-linear dependencies in the model. See Example 8.3.

Avoiding ill-posedness

Avoid continuous models which are ill-posed: the solution space is degenerate, for example consists of a single point (technically, the Slater condition is not satisfied). In general, this refers to problems which are borderline between feasible and infeasible. See Example 8.1.

Scaling the expected solution

Try to formulate the problem in such a way that the expected solution (both primal and dual) is not very large. Consult the solution summary Sec. 8.1.2 to check the objective values or solution norms.

8.2.2 Further suggestions

Here are other simple suggestions that can help locate the cause of the issues. They can also be used as hints for how to tune the optimizer if fixing the root causes of the issue is not possible.

- Remove the objective and solve the feasibility problem. This can reveal issues with the objective.
- Change the objective or change the objective sense from minimization to maximization (if applicable). If the two objective values are almost identical, this may indicate that the feasible set is very small, possibly degenerate.
- Perturb the data, for instance bounds, very slightly, and compare the results.
- For linear problems: solve the problem using a different optimizer by setting the parameter *iparam.optimizer* and compare the results.
- Force the optimizer to solve the primal/dual versions of the problem by setting the parameter *iparam.intpnt_solve_form* or *iparam.sim_solve_form*. **MOSEK** has a heuristic to decide whether to dualize, but for some problems the guess is wrong an explicit choice may give better results.
- Solve the problem without presolve or some of its parts by setting the parameter *iparam.presolve_use*, see Sec. 13.1.
- Use different numbers of threads (*iparam.num_threads*) and compare the results. Very different results indicate numerical issues resulting from round-off errors.

If the problem was dumped to a file, experimenting with various parameters is facilitated with the **MOSEK** Command Line Tool or **MOSEK** Python Console Sec. 8.4.

8.2.3 Typical pitfalls

Example 8.1 (Ill-posedness)**.** A toy example of this situation is the feasibility problem

$$(x-1)^2 \leq 1,\ (x+1)^2 \leq 1$$

whose only solution is $x = 0$ and moreover replacing any 1 on the right hand side by $1 - \varepsilon$ makes the problem infeasible and replacing it by $1 + \varepsilon$ yields a problem whose solution set is an interval (fully-dimensional). This is an example of ill-posedness.

Example 8.2 (Huge solution)**.** If the norm of the expected solution is very large it may lead to numerical issues or infeasibility. For example the problem

$$(10^{-4}, x, 10^3) \in \mathcal{Q}_r^3$$

may be declared infeasible because the expected solution must satisfy $x \geq 5 \cdot 10^9$.

Example 8.3 (Near linear dependency)**.** Consider the following problem:

$$\begin{array}{lrcrcrcrcl} \text{minimize} & & & & & & & & & \\ \text{subject to} & x_1 & + & x_2 & & & & & = & 1, \\ & & & & & x_3 & + & x_4 & = & 1, \\ & - & x_1 & & - & x_3 & & & = & -1+\varepsilon, \\ & & & - & x_2 & & & - & x_4 & = & -1, \\ & & x_1, & & & x_2, & & x_3, & & x_4 & \geq & 0. \end{array}$$

If we add the equalities together we obtain:

$$0 = \varepsilon$$

which is infeasible for any $\varepsilon \neq 0$. Here infeasibility is caused by a linear dependency in the constraint matrix coupled with a precision error represented by the ε. Indeed if a problem contains linear dependencies then the problem is either infeasible or contains redundant constraints. In the above case any of the equality constraints can be removed while not changing the set of feasible solutions. To summarize linear dependencies in the constraints can give rise to infeasible problems and therefore it is better to avoid them.

Example 8.4 (Presolving very tight bounds)**.** Next consider the problem

$$\begin{array}{lrcr} \text{minimize} & & & \\ \text{subject to} & x_1 - 0.01x_2 & = & 0, \\ & x_2 - 0.01x_3 & = & 0, \\ & x_3 - 0.01x_4 & = & 0, \\ & x_1 & \geq & -10^{-9}, \\ & x_1 & \leq & 10^{-9}, \\ & x_4 & \geq & 10^{-4}. \end{array}$$

Now the **MOSEK** presolve will, for the sake of efficiency, fix variables (and constraints) that have tight bounds where tightness is controlled by the parameter *dparam.presolve_tol_x*. Since the bounds

$$-10^{-9} \leq x_1 \leq 10^{-9}$$

are tight, presolve will set $x_1 = 0$. It easy to see that this implies $x_4 = 0$, which leads to the incorrect conclusion that the problem is infeasible. However a tiny change of the value 10^{-9} makes the problem feasible. In general it is recommended to avoid ill-posed problems, but if that is not possible then one solution is to reduce parameters such as *dparam.presolve_tol_x* to say 10^{-10}. This will at least make sure that presolve does not make the undesired reduction.

8.3 Debugging infeasibility

When solving an optimization problem one typically expects to get an optimal solution, but in some cases, either by design, or (most frequently) due to an error in the formulation, the problem may become infeasible (have no solution at all).

This section

- describes the intuitions behind infeasibility,
- helps to debug (unexpectedly) infeasible problems using the command line tool and by inspecting infeasibility reports and problem data by hand,
- gives some hints for how to modify the formulation to identify the reasons for infeasibility.

If, instead, you want to fetch an infeasibility certificate directly using Optimizer API for Python, see the tutorial in Sec. 6.13.

An infeasibility certificate is only available for continuous problems, however the hints in Sec. 8.3.4 apply to a large extent also to mixed-integer problems.

8.3.1 Numerical issues

Infeasible problem status may be just an artifact of numerical issues appearing when the problem is badly-scaled, barely feasible or otherwise ill-conditioned so that it is unstable under small perturbations of the data or round-off errors. This may be visible in the solution summary if the infeasibility certificate has poor quality. See Sec. 8.1.2 for how to diagnose that and Sec. 8.2 for possible hints. Sec. 8.2.3 contains examples of situations which may lead to infeasibility for numerical reasons.

We refer to Sec. 8.2 for further information on dealing with those sort of issues. For the rest of this section we concentrate on the case when the solution summary leaves little doubt that the problem solved by the optimizer actually is infeasible.

8.3.2 Locating primal infeasibility

As an example of a primal infeasible problem consider minimizing the cost of transportation between a number of production plants and stores: Each plant produces a fixed number of goods, and each store has a fixed demand that must be met. Supply, demand and cost of transportation per unit are given in Fig. 8.1.

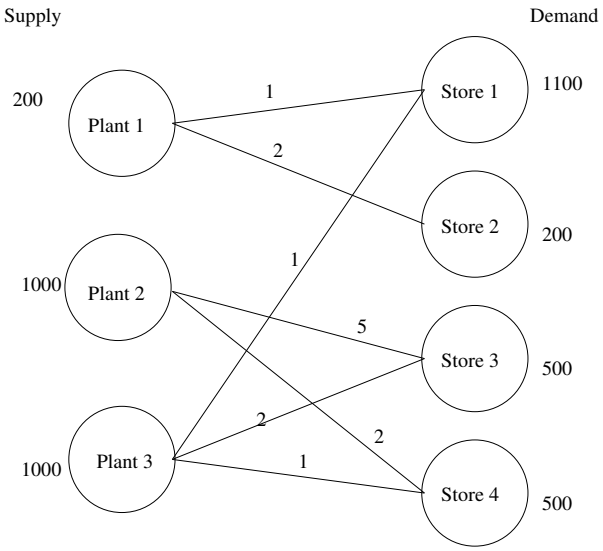

Fig. 8.1: Supply, demand and cost of transportation.

The problem represented in Fig. 8.1 is infeasible, since the total demand

$$2300 = 1100 + 200 + 500 + 500$$

exceeds the total supply

$$2200 = 200 + 1000 + 1000$$

If we denote the number of transported goods from plant i to store j by x_{ij}, the problem can be

formulated as the LP:

$$\begin{array}{lllllllllllllllll}
\text{minimize} & & x_{11} & + & 2x_{12} & + & 5x_{23} & + & 2x_{24} & + & x_{31} & + & 2x_{33} & + & x_{34} & & \\
\text{subject to} & s_0: & x_{11} & + & x_{12} & & & & & & & & & & & \leq & 200, \\
& s_1: & & & & & x_{23} & + & x_{24} & & & & & & & \leq & 1000, \\
& s_2: & & & & & & & & & x_{31} & + & x_{33} & + & x_{34} & \leq & 1000, \\
& d_0: & x_{11} & & & & & & & + & x_{31} & & & & & = & 1100, \\
& d_1: & & & x_{12} & & & & & & & & & & & = & 200, \\
& d_2: & & & & & x_{23} & + & & & & & x_{33} & & & = & 500, \\
& d_3: & & & & & & & x_{24} & + & & & & & x_{34} & = & 500, \\
& & & & & & & & & & & & & & x_{ij} & \geq & 0.
\end{array} \tag{8.1}$$

Solving problem (8.1) using **MOSEK** will result in an infeasibility status. The infeasibility certificate is contained in the dual variables an can be accessed from an API. The variables and constraints with nonzero solution values form an infeasible subproblem, which frequently is very small. See Sec. 12.1.2 or Sec. 12.2.2 for detailed specifications of infeasibility certificates.

A short infeasibility report can also be printed to the log stream. It can be turned on by setting the parameter *iparam.infeas_report_auto* to *onoffkey.on*. This causes **MOSEK** to print a report on variables and constraints which are involved in infeasibility in the above sense, i.e. have nonzero values in the certificate. The parameter *iparam.infeas_report_level* controls the amount of information presented in the infeasibility report. The default value is 1. For the above example the report is

```
Primal infeasibility report

Problem status: The problem is primal infeasible


The following constraints are involved in the primal infeasibility.


Index        Name     Lower bound      Upper bound      Dual lower       Dual upper
0            s0       none             200              0                1
2            s2       none             1000             0                1
3            d0       1100             1100             1                0
4            d1       200              200              1                0


The following bound constraints are involved in the primal infeasibility.


Index        Name     Lower bound      Upper bound      Dual lower       Dual upper
5            x33      0                none             1                0
6            x34      0                none             1                0
```

The infeasibility report is divided into two sections corresponding to constraints and variables. It is a selection of those lines from the problem solution which are important in understanding primal infeasibility. In this case the constraints `s0`, `s2`, `d0`, `d1` and variables `x33`, `x34` are of importance because of nonzero dual values. The columns `Dual lower` and `Dual upper` contain the values of dual variables s_l^c, s_u^c, s_l^x and s_u^x in the primal infeasibility certificate (see Sec. 12.1.2).

In our example the certificate means that an appropriate linear combination of constraints `s0`, `s1` with coefficient $s_u^c = 1$, constraints `d0` and `d1` with coefficient $s_u^c - s_l^c = 0 - 1 = -1$ and lower bounds on `x33` and `x34` with coefficient $-s_l^x = -1$ gives a contradiction. Indeed, the combination of the four involved constraints is $x_{33} + x_{34} \leq -100$ (as indicated in the introduction, the difference between supply and demand).

It is also possible to extract the infeasible subproblem with the command-line tool. For an infeasible problem called `infeas.lp` the command:

```
mosek -d MSK_IPAR_INFEAS_REPORT_AUTO MSK_ON infeas.lp -info rinfeas.lp
```

will produce the file `rinfeas.bas.inf.lp` which contains the infeasible subproblem. Because of its size it may be easier to work with than the original problem file.

Returning to the transportation example, we discover that removing the fifth constraint $x_{12} = 200$ makes the problem feasible. Almost all undesired infeasibilities should be fixable at the modeling stage.

8.3.3 Locating dual infeasibility

A problem may also be *dual infeasible*. In this case the primal problem is usually unbounded, meaning that feasible solutions exists such that the objective tends towards infinity. For example, consider the problem

$$\begin{array}{ll} \text{maximize} & 200y_1 + 1000y_2 + 1000y_3 + 1100y_4 + 200y_5 + 500y_6 + 500y_7 \\ \text{subject to} & y_1 + y_4 \leq 1,\ y_1 + y_5 \leq 2,\ y_2 + y_6 \leq 5,\ y_2 + y_7 \leq 2, \\ & y_3 + y_4 \leq 1,\ y_3 + y_6 \leq 2,\ y_3 + y_7 \leq 1 \\ & y_1, y_2, y_3 \leq 0 \end{array}$$

which is dual to (8.1) (and therefore is dual infeasible). The dual infeasibility report may look as follows:

```
Dual infeasibility report

Problem status: The problem is dual infeasible


The following constraints are involved in the dual infeasibility.


Index        Name     Activity         Objective        Lower bound      Upper bound
5            x33      -1                                none             2
6            x34      -1                                none             1


The following variables are involved in the dual infeasibility.


Index        Name     Activity         Objective        Lower bound      Upper bound
0            y1       -1               200              none             0
2            y3       -1               1000             none             0
3            y4       1                1100             none             none
4            y5       1                200              none             none
```

In the report we see that the variables `y1`, `y3`, `y4`, `y5` and two constraints contribute to infeasibility with non-zero values in the `Activity` column. Therefore

$$(y_1, \ldots, y_7) = (-1, 0, -1, 1, 1, 0, 0)$$

is the dual infeasibility certificate as in Sec. 12.1.2. This just means, that along the ray

$$(0,0,0,0,0,0,0) + t(y_1, \ldots, y_7) = (-t, 0, -t, t, t, 0, 0),\ t > 0,$$

which belongs to the feasible set, the objective value $100t$ can be arbitrarily large, i.e. the problem is unbounded.

In the example problem we could

- Add a lower bound on `y3`. This will directly invalidate the certificate of dual infeasibility.
- Increase the objective coefficient of `y3`. Changing the coefficients sufficiently will invalidate the inequality $c^T y^* > 0$ and thus the certificate.

8.3.4 Suggestions

Primal infeasibility

When trying to understand what causes the unexpected primal infeasible status use the following hints:

- Remove the objective function. This does not change the infeasibility status but simplifies the problem, eliminating any possibility of issues related to the objective function.
- Remove cones, semidefinite variables and integer constraints. Solve only the linear part of the problem. Typical simple modeling errors will lead to infeasibility already at this stage.
- Consider whether your problem has some obvious necessary conditions for feasibility and examine if these are satisfied, e.g. total supply should be greater than or equal to total demand.
- Verify that coefficients and bounds are reasonably sized in your problem.
- See if there are any obvious contradictions, for instance a variable is bounded both in the variables and constraints section, and the bounds are contradictory.
- Consider replacing suspicious equality constraints by inequalities. For instance, instead of $x_{12} = 200$ see what happens for $x_{12} \geq 200$ or $x_{12} \leq 200$.
- Relax bounds of the suspicious constraints or variables.
- For integer problems, remove integrality constraints on some/all variables and see if the problem solves.
- Remember that variables without explicitly initialized bounds are fixed at zero.
- Form an **elastic model**: allow to violate constraints at a cost. Introduce slack variables and add them to the objective as penalty. For instance, suppose we have a constraint

$$\begin{array}{ll} \text{minimize} & c^T x, \\ \text{subject to} & a^T x \leq b. \end{array}$$

  which might be causing infeasibility. Then create a new variable y and form the problem which contains:

$$\begin{array}{ll} \text{minimize} & c^T x + y, \\ \text{subject to} & a^T x \leq b + y. \end{array}$$

  Solving this problem will reveal by how much the constraint needs to be relaxed in order to become feasible. This is equivalent to inspecting the infeasibility certificate but may be more intuitive.
- If you think you have a feasible solution or its part, fix all or some of the variables to those values. Presolve will propagate them through the model and potentially reveal more localized sources of infeasibility.
- Dump the problem in PTF or LP format and verify that the problem that was passed to the optimizer corresponds to the problem expressed in the high-level modeling language, if any such was used.

Dual infeasibility

When trying to understand what causes the unexpected dual infeasible status use the following hints:

- Verify that the objective coefficients are reasonably sized.
- Check if no bounds and constraints are missing, for example if all variables that should be nonnegative have been declared as such etc.
- Strengthen bounds of the suspicious constraints or variables.
- Remember that constraints without explicitly initialized bounds are free (no bound).

- Form an series of models with decreasing bounds on the objective, that is, instead of objective

$$\text{minimize } c^T x$$

  solve the problem with an additional constraint such as

$$c^T x = -10^5$$

  and inspect the solution to figure out the mechanism behind arbitrarily decreasing objective values. This is equivalent to inspecting the infeasibility certificate but may be more intuitive.

- Dump the problem in PTF or LP format and verify that the problem that was passed to the optimizer corresponds to the problem expressed in the high-level modeling language, if any such was used.

Please note that modifying the problem to invalidate the reported certificate does *not* imply that the problem becomes feasible — the reason for infeasibility may simply *move*, resulting a problem that is still infeasible, but for a different reason. More often, the reported certificate can be used to give a hint about errors or inconsistencies in the model that produced the problem.

8.4 Python Console

The **MOSEK** Python Console is an alternative to the **MOSEK** Command Line Tool. It can be used for interactive loading, solving and debugging optimization problems stored in files, for example **MOSEK** task files. It facilitates debugging techniques described in Sec. 8.

8.4.1 Usage

The tool requires Python 3. The **MOSEK** interface for Python must be installed following the installation instructions for Python API or Python Fusion API. The easiest option is

```
pip install Mosek
```

The Python Console is contained in the file `mosekconsole.py` in the folder with **MOSEK** binaries. It can be copied to an arbitrary location. The file is also available for `download here (mosekconsole.py)`.

To run the console in interactive mode use

```
python mosekconsole.py
```

To run the console in batch mode provide a semicolon-separated list of commands as the second argument of the script, for example:

```
python mosekconsole.py "read data.task.gz; solve form=dual; writesol data"
```

The script is written using the **MOSEK** Python API and can be extended by the user if more specific functionality is required. We refer to the documentation of the Python API.

8.4.2 Examples

To read a problem from `data.task.gz`, solve it, and write solutions to `data.sol`, `data.bas` or `data.itg`:

```
read data.task.gz; solve; writesol data
```

To convert between file formats:

```
read data.task.gz; write data.mps
```

To set a parameter before solving:

```
read data.task.gz; param INTPNT_CO_TOL_DFEAS 1e-9; solve"
```

To list parameter values related to the mixed-integer optimizer in the task file:

```
read data.task.gz; param MIO
```

To print a summary of problem structure:

```
read data.task.gz; anapro
```

To solve a problem forcing the dual and switching off presolve:

```
read data.task.gz; solve form=dual presolve=no
```

To write an infeasible subproblem to a file for debugging purposes:

```
read data.task.gz; solve; infsub; write inf.opf
```

8.4.3 Full list of commands

Below is a brief description of all the available commands. Detailed information about a specific command `cmd` and its options can be obtained with

```
help cmd
```

Table 8.1: List of commands of the MOSEK Python Console.

Command	Description
`help [command]`	Print list of commands or info about a specific command
`log filename`	Save the session to a file
`intro`	Print MOSEK splashscreen
`testlic`	Test the license system
`read filename`	Load problem from file
`reread`	Reload last problem file
`solve [options]`	Solve current problem
`write filename`	Write current problem to file
`param [name [value]]`	Set a parameter or get parameter values
`paramdef`	Set all parameters to default values
`paramdiff`	Show parameters with non-default values
`paramval name`	Show available values for a parameter
`info [name]`	Get an information item
`anapro`	Analyze problem data
`anapro+`	Analyze problem data with the internal analyzer
`hist`	Plot a histogram of problem data
`histsol`	Plot a histogram of the solutions
`spy`	Plot the sparsity pattern of the data matrices
`truncate epsilon`	Truncate small coefficients down to 0
`resobj [fac]`	Rescale objective by a factor
`rlb thr`	Remove large bounds
`anasol`	Analyze solutions
`removeitg`	Remove integrality constraints
`removecones`	Remove all cones and leave just the linear part
`delsol`	Remove solutions
`fixsol solname`	Fix all variables to a specific solution
`fixintsol`	Fix all integer variables to a specific solution
`infsub`	Replace current problem with its infeasible subproblem
`dualize`	Replace current problem with its dual
`writesol basename`	Write solution(s) to file(s) with given basename

continues on next page

Table 8.1 – continued from previous page

Command	Description
`writejsonsol name`	Write solutions to JSON file with given name
`ptf`	Print the PTF representation of the problem
`optserver [url]`	Use an OptServer to optimize
`ls`	List the current folder
`exit`	Leave

Chapter 9

Advanced Numerical Tutorials

9.1 Solving Linear Systems Involving the Basis Matrix

A linear optimization problem always has an optimal solution which is also a basic solution. In an optimal basic solution there are exactly m basic variables where m is the number of rows in the constraint matrix A. Define

$$B \in \mathbb{R}^{m \times m}$$

as a matrix consisting of the columns of A corresponding to the basic variables. The basis matrix B is always non-singular, i.e.

$$\det(B) \neq 0$$

or, equivalently, B^{-1} exists. This implies that the linear systems

$$B\bar{x} = w \tag{9.1}$$

and

$$B^T\bar{x} = w \tag{9.2}$$

each have a unique solution for all w.

MOSEK provides functions for solving the linear systems (9.1) and (9.2) for an arbitrary w. In the next sections we will show how to use **MOSEK** to

- *identify the solution basis*,
- *solve arbitrary linear systems*.

9.1.1 Basis identification

To use the solutions to (9.1) and (9.2) it is important to know how the basis matrix B is constructed.

Internally **MOSEK** employs the linear optimization problem

$$\begin{array}{lrcccl} \text{maximize} & & & c^T x & & \\ \text{subject to} & & & Ax - x^c & = & 0, \\ & l^x & \leq & x & \leq & u^x, \\ & l^c & \leq & x^c & \leq & u^c. \end{array} \tag{9.3}$$

where

$$x^c \in \mathbb{R}^m \text{ and } x \in \mathbb{R}^n.$$

The basis matrix is constructed of m columns taken from

$$\left[\begin{array}{cc} A & -I \end{array} \right].$$

If variable x_j is a basis variable, then the j-th column of A, denoted $a_{:,j}$, will appear in B. Similarly, if x_i^c is a basis variable, then the i-th column of $-I$ will appear in the basis. The ordering of the basis variables and therefore the ordering of the columns of B is arbitrary. The ordering of the basis variables may be retrieved by calling the function *Task.initbasissolve*. This function initializes data structures for later use and returns the indexes of the basic variables in the array `basis`. The interpretation of the `basis` is as follows. If we have

$$\texttt{basis}[i] < \texttt{numcon}$$

then the i-th basis variable is

$$x^c_{\texttt{basis}[i]}.$$

Moreover, the i-th column in B will be the i-th column of $-I$. On the other hand if

$$\texttt{basis}[i] \geq \texttt{numcon},$$

then the i-th basis variable is the variable

$$x_{\texttt{basis}[i]-\texttt{numcon}}$$

and the i-th column of B is the column

$$A_{:,(\texttt{basis}[i]-\texttt{numcon})}.$$

For instance if $\texttt{basis}[0] = 4$ and $\texttt{numcon} = 5$, then since $\texttt{basis}[0] < \texttt{numcon}$, the first basis variable is x_4^c. Therefore, the first column of B is the fourth column of $-I$. Similarly, if $\texttt{basis}[1] = 7$, then the second variable in the basis is $x_{\texttt{basis}[1]-\texttt{numcon}} = x_2$. Hence, the second column of B is identical to $a_{:,2}$.

An example

Consider the linear optimization problem:

$$\begin{array}{lrcl} \text{minimize} & x_0 + x_1 & & \\ \text{subject to} & x_0 + 2x_1 & \leq & 2, \\ & x_0 + x_1 & \leq & 6, \\ & x_0, x_1 \geq 0. & & \end{array} \tag{9.4}$$

Suppose a call to *Task.initbasissolve* returns an array `basis` so that

```
basis[0] = 1,
basis[1] = 2.
```

Then the basis variables are x_1^c and x_0 and the corresponding basis matrix B is

$$\begin{bmatrix} 0 & 1 \\ -1 & 1 \end{bmatrix}.$$

Please note the ordering of the columns in B .

Listing 9.1: A program showing how to identify the basis.

```
import mosek
import sys

def streamprinter(text):
    sys.stdout.write(text)
    sys.stdout.flush()

def main():
    numcon = 2
```

(continues on next page)

(continued from previous page)

```
    numvar = 2

    # Since the value infinity is never used, we define
    # 'infinity' symbolic purposes only
    infinity = 0

    c = [1.0, 1.0]
    ptrb = [0, 2]
    ptre = [2, 3]
    asub = [0, 1,
            0, 1]
    aval = [1.0, 1.0,
            2.0, 1.0]
    bkc = [mosek.boundkey.up,
           mosek.boundkey.up]

    blc = [-infinity,
           -infinity]
    buc = [2.0,
           6.0]

    bkx = [mosek.boundkey.lo,
           mosek.boundkey.lo]
    blx = [0.0,
           0.0]

    bux = [+infinity,
           +infinity]
    w1 = [2.0, 6.0]
    w2 = [1.0, 0.0]

    try:
        with mosek.Task() as task:
            task.set_Stream(mosek.streamtype.log, streamprinter)
            task.inputdata(numcon, numvar,
                           c,
                           0.0,
                           ptrb,
                           ptre,
                           asub,
                           aval,
                           bkc,
                           blc,
                           buc,
                           bkx,
                           blx,
                           bux)
            task.putobjsense(mosek.objsense.maximize)
            r = task.optimize()
            if r != mosek.rescode.ok:
                print("Mosek warning:", r)

            basis = task.initbasissolve()

            #List basis variables corresponding to columns of B
            varsub = [0, 1]
```

(continues on next page)

(continued from previous page)

```
            for i in range(numcon):
                if basis[varsub[i]] < numcon:
                    print("Basis variable no %d is xc%d" % (i, basis[i]))
                else:
                    print("Basis variable no %d is x%d" %
                          (i, basis[i] - numcon))

            # solve Bx = w1
            # varsub contains index of non-zeros in b.
            #  On return b contains the solution x and
            # varsub the index of the non-zeros in x.
            nz = 2

            nz = task.solvewithbasis(False, nz, varsub, w1)
            print("nz = %s" % nz)
            print("Solution to Bx = w1:")

            for i in range(nz):
                if basis[varsub[i]] < numcon:
                    print("xc %s = %s" % (basis[varsub[i]], w1[varsub[i]]))
                else:
                    print("x%s = %s" %
                          (basis[varsub[i]] - numcon, w1[varsub[i]]))

            # Solve B^Tx = w2
            nz = 1
            varsub[0] = 0

            nz = task.solvewithbasis(True, nz, varsub, w2)

            print("Solution to B^Tx = w2:")

            for i in range(nz):
                if basis[varsub[i]] < numcon:
                    print("xc %s = %s" % (basis[varsub[i]], w2[varsub[i]]))
                else:
                    print("x %s = %s" %
                          (basis[varsub[i]] - numcon, w2[varsub[i]]))
    except Exception as e:
        print(e)

if __name__ == '__main__':
    main()
```

In the example above the linear system is solved using the optimal basis for (9.4) and the original right-hand side of the problem. Thus the solution to the linear system is the optimal solution to the problem. When running the example program the following output is produced.

```
basis[0] = 1
Basis variable no 0 is xc1.
basis[1] = 2
Basis variable no 1 is x0.

Solution to Bx = b:

x0 = 2.000000e+00
```

(continues on next page)

(continued from previous page)

```
xc1 = -4.000000e+00

Solution to B^Tx = c:

x1 = -1.000000e+00
x0 = 1.000000e+00
```

Please note that the ordering of the basis variables is

$$\begin{bmatrix} x_1^c \\ x_0 \end{bmatrix}$$

and thus the basis is given by:

$$B = \begin{bmatrix} 0 & 1 \\ -1 & 1 \end{bmatrix}$$

It can be verified that

$$\begin{bmatrix} x_1^c \\ x_0 \end{bmatrix} = \begin{bmatrix} -4 \\ 2 \end{bmatrix}$$

is a solution to

$$\begin{bmatrix} 0 & 1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} x_1^c \\ x_0 \end{bmatrix} = \begin{bmatrix} 2 \\ 6 \end{bmatrix}.$$

9.1.2 Solving arbitrary linear systems

MOSEK can be used to solve an arbitrary (rectangular) linear system

$$Ax = b$$

using the *Task.solvewithbasis* function without optimizing the problem as in the previous example. This is done by setting up an A matrix in the task, setting all variables to basic and calling the *Task.solvewithbasis* function with the b vector as input. The solution is returned by the function.

An example

Below we demonstrate how to solve the linear system

$$\begin{bmatrix} 0 & 1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \tag{9.5}$$

with two inputs $b = (1, -2)$ and $b = (7, 0)$.

```
import mosek

def setup(task,
          aval,
          asub,
          ptrb,
          ptre,
          numvar,
          basis):
    # Since the value infinity is never used, we define
    # 'infinity' symbolic purposes only
    infinity = 0

    skx = [mosek.stakey.bas] * numvar
```

(continues on next page)

(continued from previous page)

```
    skc = [mosek.stakey.fix] * numvar

    task.appendvars(numvar)
    task.appendcons(numvar)

    for i in range(len(asub)):
        task.putacol(i, asub[i], aval[i])

    for i in range(numvar):
        task.putconbound(i, mosek.boundkey.fx, 0.0, 0.0)

    for i in range(numvar):
        task.putvarbound(i,
                         mosek.boundkey.fr,
                         -infinity,
                         infinity)

    # Define a basic solution by specifying
    # status keys for variables & constraints.
    task.deletesolution(mosek.soltype.bas);

    task.putskcslice(mosek.soltype.bas, 0, numvar, skc);
    task.putskxslice(mosek.soltype.bas, 0, numvar, skx);

    task.initbasissolve(basis);


def main():
    numcon = 2
    numvar = 2

    aval = [[-1.0],
            [1.0, 1.0]]
    asub = [[1],
            [0, 1]]

    ptrb = [0, 1]
    ptre = [1, 3]

    with mosek.Task() as task:
        # Directs the log task stream to the user specified
        # method task_msg_obj.streamCB
        task.set_Stream(mosek.streamtype.log,
                        lambda msg: sys.stdout.write(msg))
        # Put A matrix and factor A.
        # Call this function only once for a given task.

        basis = [0] * numvar
        b = [0.0, -2.0]
        bsub = [0, 1]

        setup(task,
              aval,
              asub,
              ptrb,
              ptre,
```

(continues on next page)

(continued from previous page)

```
                numvar,
                basis)

        # now solve rhs
        b = [1, -2]
        bsub = [0, 1]
        nz = task.solvewithbasis(False, 2, bsub, b)
        print("\nSolution to Bx = b:\n")

        # Print solution and show correspondents
        # to original variables in the problem
        for i in range(nz):
            if basis[bsub[i]] < numcon:
                print("This should never happen")
            else:
                print("x%d = %d" % (basis[bsub[i]] - numcon, b[bsub[i]]))

        b[0] = 7
        bsub[0] = 0

        nz = task.solvewithbasis(False, 1, bsub, b)

        print("\nSolution to Bx = b:\n")
        # Print solution and show correspondents
        # to original variables in the problem
        for i in range(nz):
            if basis[bsub[i]] < numcon:
                print("This should never happen")
            else:
                print("x%d = %d" % (basis[bsub[i]] - numcon, b[bsub[i]]))

if __name__ == "__main__":
    try:
        main()
    except:
        import traceback
        traceback.print_exc()
```

The most important step in the above example is the definition of the basic solution, where we define the status key for each variable. The actual values of the variables are not important and can be selected arbitrarily, so we set them to zero. All variables corresponding to columns in the linear system we want to solve are set to basic and the slack variables for the constraints, which are all non-basic, are set to their bound.

The program produces the output:

```
Solution to Bx = b:

x1 = 1
x0 = 3

Solution to Bx = b:

x1 = 7
x0 = 7
```

9.2 Calling BLAS/LAPACK Routines from MOSEK

Sometimes users need to perform linear algebra operations that involve dense matrices and vectors. Also **MOSEK** extensively uses high-performance linear algebra routines from the BLAS and LAPACK packages and some of these routines are included in the package shipped to the users.

The **MOSEK** versions of BLAS/LAPACK routines:

- use **MOSEK** data types and return value conventions,
- preserve the BLAS/LAPACK naming convention.

Therefore the user can leverage on efficient linear algebra routines, with a simplified interface, with no need for additional packages.

List of available routines

Table 9.1: BLAS routines available.

BLAS Name	**MOSEK** function	Math Expression
AXPY	*Env.axpy*	$y = \alpha x + y$
DOT	*Env.dot*	$x^T y$
GEMV	*Env.gemv*	$y = \alpha Ax + \beta y$
GEMM	*Env.gemm*	$C = \alpha AB + \beta C$
SYRK	*Env.syrk*	$C = \alpha AA^T + \beta C$

Table 9.2: LAPACK routines available.

LAPACK Name	**MOSEK** function	Description
POTRF	*Env.potrf*	Cholesky factorization of a semidefinite symmetric matrix
SYEVD	*Env.syevd*	Eigenvalues and eigenvectors of a symmetric matrix
SYEIG	*Env.syeig*	Eigenvalues of a symmetric matrix

Source code examples

In Listing 9.2 we provide a simple working example. It has no practical meaning except showing how to organize the input and call the methods.

Listing 9.2: Calling BLAS and LAPACK routines from Optimizer API for Python.

```
import mosek

def print_matrix(x, r, c):
    for i in range(r):
        print([x[j * r + i] for j in range(c)])

with mosek.Env() as env:

    n = 3
    m = 2
    k = 3

    alpha = 2.0
    beta = 0.5

    x = [1.0, 1.0, 1.0]
    y = [1.0, 2.0, 3.0]
    z = [1.0, 1.0]
```

(continues on next page)

(continued from previous page)

```
    v = [0.0, 0.0]
    #A has m=2 rows and k=3 cols
    A = [1.0, 1.0, 2.0, 2.0, 3., 3.]
    #B has k=3 rows and n=3 cols
    B = [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
    C = [0.0 for i in range(n * m)]
    D = [1.0, 1.0, 1.0, 1.0]
    Q = [1.0, 0.0, 0.0, 2.0]

# BLAS routines

    xy = env.dot(n, x, y)
    print("dot results= %f\n" % xy)

    env.axpy(n, alpha, x, y)
    print("\naxpy results is ")
    print_matrix(y, 1, len(y))

    env.gemv(mosek.transpose.no, m, n, alpha, A, x, beta, z)
    print("\ngemv results is ")
    print_matrix(z, 1, len(z))

    env.gemm(mosek.transpose.no, mosek.transpose.no,
             m, n, k, alpha, A, B, beta, C)
    print("\ngemm results is ")
    print_matrix(C, m, n)

    env.syrk(mosek.uplo.lo, mosek.transpose.no, m, k, alpha, A, beta, D)
    print("\nsyrk results is")
    print_matrix(D, m, m)

# LAPACK routines

    env.potrf(mosek.uplo.lo, m, Q)
    print("\npotrf results is ")
    print_matrix(Q, m, m)


    env.syeig(mosek.uplo.lo, m, Q, v)
    print("\nsyeig results is")
    print_matrix(v, 1, m)


    env.syevd(mosek.uplo.lo, m, Q, v)
    print("\nsyevd results is")
    print('v: ')
    print_matrix(v, 1, m)
    print('Q: ')
    print_matrix(Q, m, m)

    print("Exiting...")
```

9.3 Computing a Sparse Cholesky Factorization

Given a positive semidefinite symmetric (PSD) matrix

$$A \in \mathbb{R}^{n \times n}$$

it is well known there exists a matrix L such that

$$A = LL^T.$$

If the matrix L is lower triangular then it is called a *Cholesky factorization*. Given A is positive definite (nonsingular) then L is also nonsingular. A Cholesky factorization is useful for many reasons:

- A system of linear equations $Ax = b$ can be solved by first solving the lower triangular system $Ly = b$ followed by the upper triangular system $L^T x = y$.
- A quadratic term $x^T Ax$ in a constraint or objective can be replaced with $y^T y$ for $y = L^T x$, potentially leading to a more robust formulation (see [And13]).

Therefore, **MOSEK** provides a function that can compute a Cholesky factorization of a PSD matrix. In addition a function for solving linear systems with a nonsingular lower or upper triangular matrix is available.

In practice A may be very large with n is in the range of millions. However, then A is typically sparse which means that most of the elements in A are zero, and sparsity can be exploited to reduce the cost of computing the Cholesky factorization. The computational savings depend on the positions of zeros in A. For example, below a matrix A is given together with a Cholesky factor up to 5 digits of accuracy:

$$A = \begin{bmatrix} 4 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}, \quad L = \begin{bmatrix} 2.0000 & 0 & 0 & 0 \\ 0.5000 & 0.8660 & 0 & 0 \\ 0.5000 & -0.2887 & 0.8165 & 0 \\ 0.5000 & -0.2887 & -0.4082 & 0.7071 \end{bmatrix}. \tag{9.6}$$

However, if we symmetrically permute the rows and columns of A using a permutation matrix P

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}, \quad A' = PAP^T = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 4 \end{bmatrix},$$

then the Cholesky factorization of $A' = L'L'^T$ is

$$L' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

which is sparser than L.

Computing a permutation matrix that leads to the sparsest Cholesky factorization or the minimal amount of work is NP-hard. Good permutations can be chosen by using heuristics, such as the minimum degree heuristic and variants. The function *Env.computesparsecholesky* provided by **MOSEK** for computing a Cholesky factorization has a build in permutation aka. reordering heuristic. The following code illustrates the use of *Env.computesparsecholesky* and *Env.sparsetriangularsolvedense*.

Listing 9.3: How to use the sparse Cholesky factorization routine available in **MOSEK**.

```
try:
    perm, diag, lnzc, lptrc, lensubnval, lsubc, lvalc = env.computesparsecholesky(
        0,      # Mosek chooses number of threads
        1,      # Use reordering heuristic
```

(continues on next page)

(continued from previous page)

```
            1.0e-14,# Singularity tolerance
            anzc, aptrc, asubc, avalc)

        printsparse(n, perm, diag, lnzc, lptrc, lensubnval, lsubc, lvalc)

        x = [b[p] for p in perm]    # Permuted b is stored as x.

        # Compute inv(L)*x.
        env.sparsetriangularsolvedense(mosek.transpose.no,
                                       lnzc, lptrc, lsubc, lvalc, x)

        # Compute inv(L^T)*x.
        env.sparsetriangularsolvedense(mosek.transpose.yes,
                                       lnzc, lptrc, lsubc, lvalc, x)

        print("\nSolution Ax=b: x = ", numpy.array(
            [x[j] for i in range(n) for j in range(n) if perm[j] == i]), "\n")
    except:
        raise
```

We can set up the data to recreate the matrix A from (9.6):

```
    # Observe that anzc, aptrc, asubc and avalc only specify the lower
    # triangular part.
    n = 4
    anzc = [4, 1, 1, 1]
    asubc = [0, 1, 2, 3, 1, 2, 3]
    aptrc = [0, 4, 5, 6]
    avalc = [4.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
    b = [13.0, 3.0, 4.0, 5.0]
```

and we obtain the following output:

```
Example with positive definite A.
P = [ 3 2 0 1 ]
diag(D) = [ 0.00 0.00 0.00 0.00 ]
L=
1.00 0.00 0.00 0.00
0.00 1.00 0.00 0.00
1.00 1.00 1.41 0.00
0.00 0.00 0.71 0.71

Solution A x = b, x = [ 1.00 2.00 3.00 4.00 ]
```

The output indicates that with the permutation matrix

$$P = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

there is a Cholesky factorization $PAP^T = LL^T$, where

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1.4142 & 0 \\ 0 & 0 & 0.7071 & 0.7071 \end{bmatrix}$$

The remaining part of the code solvers the linear system $Ax = b$ for $b = [13, 3, 4, 5]^T$. The solution is reported to be $x = [1, 2, 3, 4]^T$, which is correct.

The second example shows what happens when we compute a sparse Cholesky factorization of a singular matrix. In this example A is a rank 1 matrix

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}^T \tag{9.7}$$

```
#Example 2 - singular A
n = 3
anzc = [3, 2, 1]
asubc = [0, 1, 2, 1, 2, 2]
aptrc = [0, 3, 5]
avalc = [1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
```

Now we get the output

```
P = [ 0 2 1 ]
diag(D) = [ 0.00e+00 1.00e-14 1.00e-14 ]
L=
1.00e+00 0.00e+00 0.00e+00
1.00e+00 1.00e-07 0.00e+00
1.00e+00 0.00e+00 1.00e-07
```

which indicates the decomposition

$$PAP^T = LL^T - D$$

where

$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \quad L = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 10^{-7} & 0 \\ 1 & 0 & 10^{-7} \end{bmatrix}, \quad D = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 10^{-14} & 0 \\ 0 & 0 & 10^{-14} \end{bmatrix}.$$

Since A is only positive semdefinite, but not of full rank, some of diagonal elements of A are boosted to make it truely positive definite. The amount of boosting is passed as an argument to *Env.computesparsecholesky*, in this case 10^{-14}. Note that

$$PAP^T = LL^T - D$$

where D is a small matrix so the computed Cholesky factorization is exact of slightly perturbed A. In general this is the best we can hope for in finite precision and when A is singular or close to being singular.

We will end this section by a word of caution. Computing a Cholesky factorization of a matrix that is not of full rank and that is not suffciently well conditioned may lead to incorrect results i.e. a matrix that is indefinite may declared positive semidefinite and vice versa.

Chapter 10

Technical guidelines

This section contains some more in-depth technical guidelines for Optimizer API for Python, not strictly necessary for basic use of **MOSEK**.

10.1 Memory management and garbage collection

Users should make sure the **MOSEK** objects are fully cleaned up before they go out of scope so that no internally allocated memory leaks. Memory leaks can manifest themselves especially as:

- memory usage not decreasing after the solver terminates,
- memory usage increasing when solving a sequence of problems.

The recommended way is to always use the task object within a *managed context*, so that the built-in garbage collector will call its internal clean-up method when the object goes out of scope. That will ensure that all memory allocated internally by the shared library will also be freed.

The *Task* class supports the *Context Manager* protocol, so it will be destroyed properly when used in a `with` statement:

```
with mosek.Task() as task:
  # Build an optimization problem
  # ...
```

If this is not possible, for example because the object is passed around through various calls, then the disposing method should be called manually before the object is abandoned. Note that the garbage collector is unable to automatically access the memory allocated internally by the shared library, therefore calling a manual cleanup method is necessary.

To release the memory manually use *Task.__del__* :

```
task.__del__()
```

10.2 Names

All elements of an optimization problem in **MOSEK** (objective, constraints, variables, etc.) can be given names. Assigning meaningful names to variables and constraints makes it much easier to understand and debug optimization problems dumped to a file. On the other hand, note that assigning names can substantially increase setup time, so it should be avoided in time-critical applications.

Names of various elements of the problem can be set and retrieved using various functions listed in the **Names** section of Sec. 15.2.

Note that file formats impose various restrictions on names, so not all names can be written verbatim to each type of file. If at least one name cannot be written to a given format then generic names and substitutions of offending characters will be used when saving to a file, resulting in a transformation of all names in the problem. See Sec. 16.

10.3 Multithreading

Thread safety

Sharing a task between threads is safe, as long as it is not accessed from more than one thread at a time. Multiple tasks can be created and used in parallel without any problems.

Parallelization

The interior-point and mixed-integer optimizers in **MOSEK** are parallelized. By default **MOSEK** will automatically select the number of threads. However, the maximum number of threads allowed can be changed by setting the parameter *iparam.num_threads* and related parameters. This should never exceed the number of cores.

The speed-up obtained when using multiple threads is highly problem and hardware dependent. We recommend experimenting with various thread numbers to determine the optimal settings. For small problems using multiple threads may be counter-productive because of the associated overhead. Note also that not all parts of the algorithm can be parallelized, so there are times when CPU utilization is only 1 even if more cores are available.

Determinism

By default the optimizer is run-to-run deterministic, which means that it will return the same answer each time it is run on the same machine with the same input, the same parameter settings (including number of threads) and no time limits.

Setting the number of threads

The number of threads the optimizer uses can be changed with the parameter *iparam.num_threads*.

The parameter *iparam.num_threads* affects only the optimizer. It may be the case that `numpy` is consuming more threads. In most cases this can be limited by setting the environment variable `MKL_NUM_THREADS`. See the `numpy` documentation for more details.

10.4 Timing

Unless otherwise mentioned all parameters, information items and log output entries in **MOSEK** which refer to time measurement are expressed in seconds of wall-clock time.

10.5 Efficiency

Although **MOSEK** is implemented to handle memory efficiently, the user may have valuable knowledge about a problem, which could be used to improve the performance of **MOSEK** This section discusses some tricks and general advice that hopefully make **MOSEK** process your problem faster.

Reduce the number of function calls and avoid input loops

For example, instead of setting the entries in the linear constraint matrix one by one (*Task.putaij*) define them all at once (*Task.putaijlist*) or in convenient large chunks (*Task.putacollist* etc.)

Use one or no environment

Share the environment between all tasks in one process. For most applications you don't need an explicit environment at all.

Read part of the solution

When fetching the solution, data has to be copied from the optimizer to the user's data structures. Instead of fetching the whole solution, consider fetching only the interesting part (see for example *Task.getxxslice* and similar).

Use exact array types

As much as possible pass arrays of the exact type required for each argument. For example, if a method takes an argument of type `int[]` (that is an array of 32-bit integers), then an array of type `numpy.array(dtype=numpy.int32)` will be used and passed to the solver directly, whereas an object of type `numpy.array(dtype=numpy.int64)` or `list` will have to be internally copied/converted to the exact correct type by the API. In most cases this is very fast, but for huge data arrays it may be advantageous to have them in the correct format from the start.

Avoiding memory fragmentation

MOSEK stores the optimization problem in internal data structures in the memory. Initially **MOSEK** will allocate structures of a certain size, and as more items are added to the problem the structures are reallocated. For large problems the same structures may be reallocated many times causing memory fragmentation. One way to avoid this is to give **MOSEK** an estimated size of your problem using the functions:

- *Task.putmaxnumvar*. Estimate for the number of variables.
- *Task.putmaxnumcon*. Estimate for the number of constraints.
- *Task.putmaxnumbarvar*. Estimate for the number of semidefinite matrix variables.
- *Task.putmaxnumanz*. Estimate for the number of non-zeros in A.
- *Task.putmaxnumqnz*. Estimate for the number of non-zeros in the quadratic terms.

None of these functions changes the problem, they only serve as hints. If the problem ends up growing larger, the estimates are automatically increased.

Do not mix `put-` and `get-` functions

MOSEK will queue `put-` requests internally until a `get-` function is called. If `put-` and `get-` calls are interleaved, the queue will have to be flushed more frequently, decreasing efficiency.

In general `get-` commands should not be called often (or at all) during problem setup.

Use the LIFO principle

When removing constraints and variables, try to use a LIFO (Last In First Out) approach. **MOSEK** can more efficiently remove constraints and variables with a high index than a small index.

An alternative to removing a constraint or a variable is to fix it at 0, and set all relevant coefficients to 0. Generally this will not have any impact on the optimization speed.

Add more constraints and variables than you need (now)

The cost of adding one constraint or one variable is about the same as adding many of them. Therefore, it may be worthwhile to add many variables instead of one. Initially fix the unused variable at zero, and then later unfix them as needed. Similarly, you can add multiple free constraints and then use them as needed.

Do not remove basic variables

When performing re-optimizations, instead of removing a basic variable it may be more efficient to fix the variable at zero and then remove it when the problem is re-optimized and it has left the basis. This makes it easier for **MOSEK** to restart the simplex optimizer.

10.6 The license system

MOSEK is a commercial product that **always** needs a valid license to work. **MOSEK** uses a third party license manager to implement license checking. The number of license tokens provided determines the number of optimizations that can be run simultaneously.

The following discussion is in practice only relevant for floating licenses (with a limited number of tokens). For uncounted license types (server, trial, personal academic and group) there is no need to monitor the license usage or check licenses back in since nothing restricts multiple processes from using the license.

By default a license token remains checked out from the first optimization until the end of the **MOSEK** session, i.e.

- a license token is checked out when *Task.optimize* is first called and
- it is returned when the process is terminated or when the **MOSEK** environment is deleted (if using an explicit environment created by the user, see. Sec. 7.1)

Calling *Task.optimize* from different threads using the same **MOSEK** environment only consumes one license token.

Starting the optimization when no license tokens are available will result in an error.

Default behaviour of the license system can be changed in several ways:

- Setting the parameter *iparam.cache_license* to *onoffkey.off* will force **MOSEK** to return the license token immediately after the optimization completed.
- Setting the license wait flag with the parameter *iparam.license_wait* will force **MOSEK** to wait until a license token becomes available instead of returning with an error. The wait time between checks can be set with *Env.putlicensewait*.
- Additional license checkouts and checkins can be performed with the functions *Env.checkinlicense* and *Env.checkoutlicense*.
- The default path to the license file can be changed with *Env.putlicensepath*. This must be done before the first optimization, further invocations will have no effect.
- Usually the license system is stopped automatically when the **MOSEK** library is unloaded. However, when the user explicitly unloads the library (using e.g. FreeLibrary), the license system must be stopped before the library is unloaded. This can be done by calling the function *Env.licensecleanup* as the last function call to **MOSEK**.

10.7 Deployment

When redistributing a Python application using the **MOSEK** Optimizer API for Python 11.2.6, the following shared libraries from the **MOSEK** `bin` folder are required:

- Linux : `libmosek64`, `libtbb`,
- Windows : `mosek64`, `tbb`,
- OSX : `libmosek64`, `libtbb`.

Furthermore, the folder `python/3/mosek` must be in the `PYTHONPATH`.

By default the **MOSEK** Python API will look for the binary libraries in the **MOSEK** module directory, i.e. the directory containing `__init__.py`. Alternatively, if the binary libraries reside in another directory, the application can pre-load the `mosekxx` library from another location before `mosek` is imported, e.g. like this

```
import ctypes ; ctypes.CDLL('my/path/to/mosekxx.dll')
```

Chapter 11

Case Studies

In this section we present some case studies in which the Optimizer API for Python is used to solve real-life applications. These examples involve some more advanced modeling skills and possibly some input data. The user is strongly recommended to first read the basic tutorials of Sec. 6 before going through these advanced case studies.

- *Portfolio Optimization*
  - **Keywords:** Markowitz model, variance, risk, efficient frontier, factor model, transaction cost, market impact cost
  - **Type:** Conic Quadratic, Power Cone, Mixed-Integer Optimization
- *Logistic regression*
  - **Keywords:** machine learning, logistic regression, classifier, log-sum-exp, softplus, regularization
  - **Type:** Exponential Cone, Quadratic Cone
- *Concurrent Optimizer*
  - **Keywords:** Concurrent optimization
  - **Type:** Linear Optimization, Mixed-Integer Optimization

11.1 Portfolio Optimization

In this section the Markowitz portfolio optimization problem and variants are implemented using Optimizer API for Python.

Familiarity with Sec. 6.2 is recommended to follow the syntax used to create affine conic constraints (ACCs) throughout all the models appearing in this case study.

- *Basic Markowitz model*
- *Efficient frontier*
- *Factor model and efficiency*
- *Market impact costs*
- *Transaction costs*
- *Cardinality constraints*

11.1.1 The Basic Model

The classical Markowitz portfolio optimization problem considers investing in n stocks or assets held over a period of time. Let x_j denote the amount invested in asset j, and assume a stochastic model where the return of the assets is a random variable r with known mean

$$\mu = \mathbf{E}r$$

and covariance

$$\Sigma = \mathbf{E}(r-\mu)(r-\mu)^T.$$

The return of the investment is also a random variable $y = r^T x$ with mean (or expected return)

$$\mathbf{E}y = \mu^T x$$

and variance

$$\mathbf{E}(y - \mathbf{E}y)^2 = x^T \Sigma x.$$

The standard deviation

$$\sqrt{x^T \Sigma x}$$

is usually associated with risk.

The problem facing the investor is to rebalance the portfolio to achieve a good compromise between risk and expected return, e.g., maximize the expected return subject to a budget constraint and an upper bound (denoted γ) on the tolerable risk. This leads to the optimization problem

$$\begin{array}{lrcl} \text{maximize} & \mu^T x & & \\ \text{subject to} & e^T x & = & w + e^T x^0, \\ & x^T \Sigma x & \leq & \gamma^2, \\ & x & \geq & 0. \end{array} \tag{11.1}$$

The variables x denote the investment i.e. x_j is the amount invested in asset j and x_j^0 is the initial holding of asset j. Finally, w is the initial amount of cash available.

A popular choice is $x^0 = 0$ and $w = 1$ because then x_j may be interpreted as the relative amount of the total portfolio that is invested in asset j.

Since e is the vector of all ones then

$$e^T x = \sum_{j=1}^{n} x_j$$

is the total investment. Clearly, the total amount invested must be equal to the initial wealth, which is

$$w + e^T x^0.$$

This leads to the first constraint

$$e^T x = w + e^T x^0.$$

The second constraint

$$x^T \Sigma x \leq \gamma^2$$

ensures that the variance, is bounded by the parameter γ^2. Therefore, γ specifies an upper bound of the standard deviation (risk) the investor is willing to undertake. Finally, the constraint

$$x_j \geq 0$$

excludes the possibility of short-selling. This constraint can of course be excluded if short-selling is allowed.

The covariance matrix Σ is positive semidefinite by definition and therefore there exist a matrix $G \in \mathbb{R}^{n \times k}$ such that

$$\Sigma = GG^T. \tag{11.2}$$

In general the choice of G is **not** unique and one possible choice of G is the Cholesky factorization of Σ. However, in many cases another choice is better for efficiency reasons as discussed in Sec. 11.1.3. For a given G we have that

$$\begin{array}{rcl} x^T \Sigma x & = & x^T G G^T x \\ & = & \left\| G^T x \right\|^2. \end{array}$$

Hence, we may write the risk constraint as

$$\gamma \geq \left\| G^T x \right\|$$

or equivalently

$$(\gamma, G^T x) \in \mathcal{Q}^{k+1},$$

where $\mathcal{Q}^{k+1}$ is the $(k+1)$-dimensional quadratic cone. Note that specifically when G is derived using Cholesky factorization, $k = n$.

Therefore, problem (11.1) can be written as

$$\begin{array}{lrcl} \text{maximize} & \mu^T x & & \\ \text{subject to} & e^T x & = & w + e^T x^0, \\ & (\gamma, G^T x) & \in & \mathcal{Q}^{k+1}, \\ & x & \geq & 0, \end{array} \tag{11.3}$$

which is a conic quadratic optimization problem that can easily be formulated and solved with Optimizer API for Python. Subsequently we will use the example data

$$\mu = \left[\begin{array}{cccccccc} 0.0720, & 0.1552, & 0.1754, & 0.0898, & 0.4290, & 0.3929, & 0.3217, & 0.1838 \end{array} \right]^T$$

and

$$\Sigma = \left[\begin{array}{cccccccc} 0.0946 & 0.0374 & 0.0349 & 0.0348 & 0.0542 & 0.0368 & 0.0321 & 0.0327 \\ 0.0374 & 0.0775 & 0.0387 & 0.0367 & 0.0382 & 0.0363 & 0.0356 & 0.0342 \\ 0.0349 & 0.0387 & 0.0624 & 0.0336 & 0.0395 & 0.0369 & 0.0338 & 0.0243 \\ 0.0348 & 0.0367 & 0.0336 & 0.0682 & 0.0402 & 0.0335 & 0.0436 & 0.0371 \\ 0.0542 & 0.0382 & 0.0395 & 0.0402 & 0.1724 & 0.0789 & 0.0700 & 0.0501 \\ 0.0368 & 0.0363 & 0.0369 & 0.0335 & 0.0789 & 0.0909 & 0.0536 & 0.0449 \\ 0.0321 & 0.0356 & 0.0338 & 0.0436 & 0.0700 & 0.0536 & 0.0965 & 0.0442 \\ 0.0327 & 0.0342 & 0.0243 & 0.0371 & 0.0501 & 0.0449 & 0.0442 & 0.0816 \end{array} \right].$$

Using Cholesky factorization, this implies

$$G^T = \left[\begin{array}{cccccccc} 0.3076 & 0.1215 & 0.1134 & 0.1133 & 0.1763 & 0.1197 & 0.1044 & 0.1064 \\ 0. & 0.2504 & 0.0995 & 0.0916 & 0.0669 & 0.0871 & 0.0917 & 0.0851 \\ 0. & 0. & 0.1991 & 0.0587 & 0.0645 & 0.0737 & 0.0647 & 0.0191 \\ 0. & 0. & 0. & 0.2088 & 0.0493 & 0.0365 & 0.0938 & 0.0774 \\ 0. & 0. & 0. & 0. & 0.3609 & 0.1257 & 0.1016 & 0.0571 \\ 0. & 0. & 0. & 0. & 0. & 0.2155 & 0.0566 & 0.0619 \\ 0. & 0. & 0. & 0. & 0. & 0. & 0.2251 & 0.0333 \\ 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0.2202 \end{array} \right]$$

In Sec. 11.1.3, we present a different way of obtaining G based on a factor model, that leads to more efficient computation.

Why a Conic Formulation?

Problem (11.1) is a convex quadratically constrained optimization problem that can be solved directly using **MOSEK**. Why then reformulate it as a conic quadratic optimization problem (11.3)? The main reason for choosing a conic model is that it is more robust and usually solves faster and more reliably. For instance it is not always easy to numerically validate that the matrix Σ in (11.1) is positive semidefinite due to the presence of rounding errors. It is also very easy to make a mistake so Σ becomes indefinite. These problems are completely eliminated in the conic formulation.

Moreover, observe the constraint

$$\left\|G^T x\right\| \leq \gamma$$

more numerically robust than

$$x^T \Sigma x \leq \gamma^2$$

for very small and very large values of γ. Indeed, if say $\gamma \approx 10^4$ then $\gamma^2 \approx 10^8$, which introduces a scaling issue in the model. Hence, using conic formulation we work with the standard deviation instead of variance, which usually gives rise to a better scaled model.

Example code

Listing 11.1 demonstrates how the basic Markowitz model (11.3) is implemented.

Listing 11.1: Code implementing problem (11.3).

```
    with mosek.Task() as task:
        task.set_Stream(mosek.streamtype.log, sys.stdout.write)

        # Holding variable x of length n
        # No other auxiliary variables are needed in this formulation
        task.appendvars(numvar)

        # Optionally we can give the variables names
        for j in range(0, n):
            task.putvarname(voff_x + j, "x[%d]" % (1 + j))

        # No short-selling in this model, all of x >= 0
        task.putvarboundsliceconst(voff_x, n, mosek.boundkey.lo, 0.0, inf)

        # One linear constraint: total budget
        task.appendcons(1)
        task.putconname(coff_bud, "budget")
        task.putaijlist([coff_bud] * n, range(voff_x, voff_x + n), [1.0] * n)
↪# e^T x
        rtemp = w + sum(x0)
        task.putconbound(coff_bud, mosek.boundkey.fx, rtemp, rtemp)     # equals w +␣
↪sum(x0)

        # Input (gamma, GTx) in the AFE (affine expression) storage
        # We need k+1 rows
        task.appendafes(k + 1)
        # The first affine expression = gamma
        task.putafeg(0, gamma)
        # The remaining k expressions comprise GT*x, we add them row by row
        # In more realisic scenarios it would be better to extract nonzeros and input␣
↪in sparse form
        for i in range(0, k):
            task.putafefrow(i + 1, range(voff_x, voff_x + n), GT[i])
```

(continues on next page)

(continued from previous page)

```
        # Input the affine conic constraint (gamma, GT*x) \in QCone
        # Add the quadratic domain of dimension k+1
        qdom = task.appendquadraticconedomain(k + 1)
        # Add the constraint
        task.appendaccseq(qdom, 0, None)
        task.putaccname(0, "risk")

        # Objective: maximize expected return mu^T x
        task.putclist(range(voff_x, voff_x + n), mu)
        task.putobjsense(mosek.objsense.maximize)

        # Dump the problem to a human readable PTF file.
        task.writedata("dump.ptf")

        # Solve the problem
        task.optimize()

        # Display solution summary for quick inspection of results.
        # In this simplified example we skip checks for problem and solution status
        task.solutionsummary(mosek.streamtype.msg)

        # Check if the interior point solution is an optimal point
        solsta = task.getsolsta(mosek.soltype.itr)
        if (solsta != mosek.solsta.optimal):
            # See https://docs.mosek.com/latest/pythonapi/accessing-solution.html␣
↪about handling solution statuses.
            raise Exception(f"Unexpected solution status: {solsta}")

        # Retrieve results
        xx     = task.getxxslice(mosek.soltype.itr, voff_x, voff_x + n)
        expret = task.getprimalobj(mosek.soltype.itr)

        print(f'Expected return: {expret:.10e} Std. deviation: {gamma:.4e}')
        np.set_printoptions(precision=4)
        print(f'Optimal portfolio: {np.array(xx)}')
```

The code is organized as follows:

- We have n optimization variables, one per each asset in the portfolio. They correspond to the variable x from (11.1) and their indices as variables in the task are from 0 to $n-1$ (inclusive).
- The linear part of the problem: budget constraint, no-short-selling bounds and the objective are added in the linear data of the task (A matrix, c vector and bounds) following the techniques introduced in the tutorial of Sec. 6.1.
- For the quadratic constraint we follow the path introduced in the tutorial of Sec. 6.2. We add the vector $(\gamma, G^T x)$ to the affine expression storage (AFE), create a quadratic domain of suitable length, and add the affine conic constraint (ACC) with the selected affine expressions. In the segment

```
        # Input the affine conic constraint (gamma, GT*x) \in QCone
        # Add the quadratic domain of dimension k+1
        qdom = task.appendquadraticconedomain(k + 1)
        # Add the constraint
        task.appendaccseq(qdom, 0, None)
```

we use *Task.appendaccseq* to append a single ACC with the quadratic domain `qdom` and with a sequence of affine expressions starting at position 0 in the AFE storage and of length equal to

the dimension of `qdom`. This is the simplest way to achieve what we need, since previously we also stored the required rows in AFE in the same order.

11.1.2 The Efficient Frontier

The portfolio computed by the Markowitz model is efficient in the sense that there is no other portfolio giving a strictly higher return for the same amount of risk. An efficient portfolio is also sometimes called a Pareto optimal portfolio. Clearly, an investor should only invest in efficient portfolios and therefore it may be relevant to present the investor with all efficient portfolios so the investor can choose the portfolio that has the desired tradeoff between return and risk.

Given a nonnegative α the problem

$$\begin{array}{ll} \text{maximize} & \mu^T x - \alpha x^T \Sigma x \\ \text{subject to} & e^T x = w + e^T x^0, \\ & x \geq 0. \end{array} \tag{11.4}$$

is one standard way to trade the expected return against penalizing variance. Note that, in contrast to the previous example, we explicitly use the variance ($\|G^T x\|_2^2$) rather than standard deviation ($\|G^T x\|_2$), therefore the conic model includes a rotated quadratic cone:

$$\begin{array}{lrcll} \text{maximize} & \mu^T x - \alpha s & & & \\ \text{subject to} & e^T x & = & w + e^T x^0, & \\ & (s, 0.5, G^T x) & \in & Q_r^{k+2} & \text{(equiv. to } s \geq \|G^T x\|_2^2 = x^T \Sigma x\text{)}, \\ & x & \geq & 0. & \end{array} \tag{11.5}$$

The parameter α specifies the tradeoff between expected return and variance. Ideally the problem (11.4) should be solved for all values $\alpha \geq 0$ but in practice it is impossible. Using the example data from Sec. 11.1.1, the optimal values of return and variance for several values of α are shown in the figure.

Example code

Listing 11.2 demonstrates how to compute the efficient portfolios for several values of α.

Listing 11.2: Code for the computation of the efficient frontier based on problem (11.4).

```
    with mosek.Task() as task:
        task.set_Stream(mosek.streamtype.log, sys.stdout.write)

        # Variables:
        task.appendvars(numvar)

        # Optionally we can give the variables names
        for j in range(0, n):
            task.putvarname(voff_x + j, "x[%d]" % (1 + j))
        task.putvarname(voff_s, "s")

        # No short-selling in this model, all of x >= 0
        task.putvarboundsliceconst(voff_x, n, mosek.boundkey.lo, 0.0, inf)

        # s is free variable
        task.putvarbound(voff_s, mosek.boundkey.fr, -inf, inf)

        # One linear constraint: total budget
        task.appendcons(1)
        task.putconname(coff_bud, "budget")
        task.putaijlist([coff_bud] * n, range(voff_x, voff_x + n), [1.0] * n)
↪# e^T x
        rtemp = w + sum(x0)
```

(continues on next page)

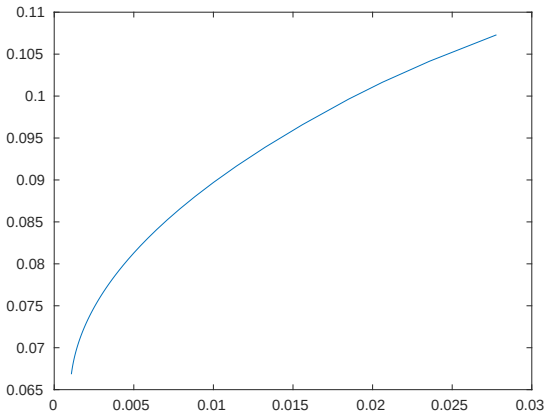

Fig. 11.1: The efficient frontier for the sample data.

(continued from previous page)

```
        task.putconbound(coff_bud, mosek.boundkey.fx, rtemp, rtemp)     # equals w +␣
↪sum(x0)

        # Input (gamma, GTx) in the AFE (affine expression) storage
        # We build the following F and g for variables [x, s]:
        #     [0, 1]      [0  ]
        # F = [0, 0], g = [0.5]
        #     [GT,0]      [0  ]
        # We need k+2 rows
        task.appendafes(k + 2)
        # The first affine expression is variable s (last variable, index n)
        task.putafefrow(0, [n], [1.0])
        # The second affine expression is constant 0.5
        task.putafeg(1, 0.5)
        # The remaining k affine expressions comprise GT*x, we add them row by row
        # In more realisic scenarios it would be better to extract nonzeros and input␣
↪in sparse form
        for i in range(0, k):
            task.putafefrow(i + 2, range(voff_x, voff_x + n), GT[i])

        # Input the affine conic constraint (s, 0.5, GT*x) \in RQCone
        # Add the quadratic domain of dimension k+1
        rqdom = task.appendrquadraticconedomain(k + 2)
        # Add the constraint
        task.appendaccseq(rqdom, 0, None)
        task.putaccname(0, "risk")

        # Set objective coefficients (x part): mu'x - alpha * s
        task.putclist(range(voff_x, voff_x + n), mu)

        task.putobjsense(mosek.objsense.maximize)

        # Turn all log output off.
        task.putintparam(mosek.iparam.log, 0)

        for alpha in alphas:
            # Dump the problem to a human readable PTF file.
            task.writedata("dump.ptf")

            task.putcj(voff_s, -alpha)

            task.optimize()

            # Display the solution summary for quick inspection of results.
            # task.solutionsummary(mosek.streamtype.msg)

            # Check if the interior point solution is an optimal point
            solsta = task.getsolsta(mosek.soltype.itr)

            if solsta in [mosek.solsta.optimal]:
                expret = 0.0
                x = task.getxxslice(mosek.soltype.itr, voff_x, voff_x + n)
                for j in range(0, n):
                    expret += mu[j] * x[j]

                stddev = np.sqrt(task.getxxslice(mosek.soltype.itr, voff_s, voff_s +␣
```

(continues on next page)

(continued from previous page)

```
↪1))

                print("alpha = {0:.2e} exp. ret. = {1:.3e}, std. dev. {2:.3e}".
↪format(alpha, expret, stddev[0]))
            else:
                # See https://docs.mosek.com/latest/pythonapi/accessing-solution.html␣
↪about handling solution statuses.
                print("An error occurred when solving for alpha=%e\n" % alpha)
                raise Exception(f"Unexpected solution status: {solsta}")
```

Note that we changed the coefficient α of the variable s in a loop. This way we were able to reuse the same model for all solves along the efficient frontier, simply changing the value of α between the solves.

11.1.3 Factor model and efficiency

In practice it is often important to solve the portfolio problem very quickly. Therefore, in this section we discuss how to improve computational efficiency at the modeling stage.

The computational cost is of course to some extent dependent on the number of constraints and variables in the optimization problem. However, in practice a more important factor is the sparsity: the number of nonzeros used to represent the problem. Indeed it is often better to focus on the number of nonzeros in G see (11.2) and try to reduce that number by for instance changing the choice of G.

In other words if the computational efficiency should be improved then it is always good idea to start with focusing at the covariance matrix. As an example assume that

$$\Sigma = D + VV^T$$

where D is a positive definite diagonal matrix. Moreover, V is a matrix with n rows and k columns. Such a model for the covariance matrix is called a factor model and usually k is much smaller than n. In practice k tends to be a small number independent of n, say less than 100.

One possible choice for G is the Cholesky factorization of Σ which requires storage proportional to $n(n+1)/2$. However, another choice is

$$G = \begin{bmatrix} D^{1/2} & V \end{bmatrix}$$

because then

$$GG^T = D + VV^T.$$

This choice requires storage proportional to $n + kn$ which is much less than for the Cholesky choice of G. Indeed assuming k is a constant storage requirements are reduced by a factor of n.

The example above exploits the so-called factor structure and demonstrates that an alternative choice of G may lead to a significant reduction in the amount of storage used to represent the problem. This will in most cases also lead to a significant reduction in the solution time.

The lesson to be learned is that it is important to investigate how the covariance matrix is formed. Given this knowledge it might be possible to make a special choice for G that helps reducing the storage requirements and enhance the computational efficiency. More details about this process can be found in [And13].

Factor model in finance

Factor model structure is typical in financial context. It is common to model security returns as the sum of two components using a factor model. The first component is the linear combination of a small number of factors common among a group of securities. The second component is a residual, specific to each security. It can be written as $R = \sum_j \beta_j F_j + \theta$, where R is a random variable representing the return of a security at a particular point in time, F_j is the random variable representing the common factor j, β_j is the exposure of the return to factor j, and θ is the specific component.

Such a model will result in the covariance structure

$$\Sigma = \Sigma_\theta + \beta \Sigma_F \beta^T,$$

where Σ_F is the covariance of the factors and Σ_θ is the residual covariance. This structure is of the form discussed earlier with $D = \Sigma_\theta$ and $V = \beta P$, assuming the decomposition $\Sigma_F = PP^T$. If the number of factors k is low and Σ_θ is diagonal, we get a very sparse G that provides the storage and solution time benefits.

Example code

Here we will work with the example data of a two-factor model ($k = 2$) built using the variables

$$\beta = \begin{bmatrix} 0.4256 & 0.1869 \\ 0.2413 & 0.3877 \\ 0.2235 & 0.3697 \\ 0.1503 & 0.4612 \\ 1.5325 & -0.2633 \\ 1.2741 & -0.2613 \\ 0.6939 & 0.2372 \\ 0.5425 & 0.2116 \end{bmatrix},$$

$$\theta = [0.0720, 0.0508, 0.0377, 0.0394, 0.0663, 0.0224, 0.0417, 0.0459],$$

and the factor covariance matrix is

$$\Sigma_F = \begin{bmatrix} 0.0620 & 0.0577 \\ 0.0577 & 0.0908 \end{bmatrix},$$

giving

$$P = \begin{bmatrix} 0.2491 & 0. \\ 0.2316 & 0.1928 \end{bmatrix}.$$

Then the matrix G would look like

$$G = \begin{bmatrix} \beta P & \Sigma_\theta^{1/2} \end{bmatrix} = \begin{bmatrix} 0.1493 & 0.0360 & 0.2683 & 0. & 0. & 0. & 0. & 0. & 0. & 0. \\ 0.1499 & 0.0747 & 0. & 0.2254 & 0. & 0. & 0. & 0. & 0. & 0. \\ 0.1413 & 0.0713 & 0. & 0. & 0.1942 & 0. & 0. & 0. & 0. & 0. \\ 0.1442 & 0.0889 & 0. & 0. & 0. & 0.1985 & 0. & 0. & 0. & 0. \\ 0.3207 & -0.0508 & 0. & 0. & 0. & 0. & 0.2576 & 0. & 0. & 0. \\ 0.2568 & -0.0504 & 0. & 0. & 0. & 0. & 0. & 0.1497 & 0. & 0. \\ 0.2277 & 0.0457 & 0. & 0. & 0. & 0. & 0. & 0. & 0.2042 & 0. \\ 0.1841 & 0.0408 & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0.2142 \end{bmatrix}.$$

This matrix is indeed very sparse.

In general, we get an $n \times (n+k)$ size matrix this way with k full columns and an $n \times n$ diagonal part. In order to maintain a sparse representation we do not construct the matrix G explicitly in the code but instead work with two pieces of data: the dense matrix $G_{\text{factor}} = \beta P$ of shape $n \times k$ and the diagonal vector θ of length n.

Example code

In the following we demonstrate how to write code to compute the matrix G_{factor} of the factor model. We start with the inputs

Listing 11.3: Inputs for the computation of the matrix G_{factor} from the factor model.

```
B = np.array([
    [0.4256, 0.1869],
    [0.2413, 0.3877],
    [0.2235, 0.3697],
    [0.1503, 0.4612],
    [1.5325, -0.2633],
```

(continues on next page)

(continued from previous page)

```
        [1.2741, -0.2613],
        [0.6939, 0.2372],
        [0.5425, 0.2116]
    ])
    S_F = np.array([
        [0.0620, 0.0577],
        [0.0577, 0.0908]
    ])
    theta = np.array([0.0720, 0.0508, 0.0377, 0.0394, 0.0663, 0.0224, 0.0417, 0.0459])
```

Then the matrix G_{factor} is obtained as:

```
    P        = np.linalg.cholesky(S_F)
    G_factor = B @ P
```

The code for computing an optimal portfolio in the factor model is very similar to the one from the basic model in Listing 11.1 with one notable exception: we construct the expression $G^T x$ appearing in the conic constraint by stacking together two separate vectors $G_{\text{factor}}^T x$ and $\Sigma_\theta^{1/2} x$:

```
        # Input (gamma, G_factor_T x, diag(sqrt(theta))*x) in the AFE (affine␣
↪expression) storage
        # We need k+n+1 rows and we fill them in in three parts
        task.appendafes(k + n + 1)
        # 1. The first affine expression = gamma, will be specified later
        # 2. The next k expressions comprise G_factor_T*x, we add them row by row
        #    transposing the matrix G_factor on the fly
        for i in range(0, k):
            task.putafefrow(i + 1, range(voff_x, voff_x + n), np.array(G_factor[:,i]))
        # 3. The remaining n rows contain sqrt(theta) on the diagonal
        task.putafefentrylist(range(k + 1, k + 1 + n), range(voff_x, voff_x + n), np.
↪sqrt(theta))
```

The full code is demonstrated below:

Listing 11.4: Implementation of portfolio optimization in the factor model.

```
import mosek
import sys
import numpy as np

if __name__ == '__main__':

    n = 8
    w = 1.0
    mu = [0.07197, 0.15518, 0.17535, 0.08981, 0.42896, 0.39292, 0.32171, 0.18379]
    x0 = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
    B = np.array([
        [0.4256, 0.1869],
        [0.2413, 0.3877],
        [0.2235, 0.3697],
        [0.1503, 0.4612],
        [1.5325, -0.2633],
        [1.2741, -0.2613],
        [0.6939, 0.2372],
        [0.5425, 0.2116]
    ])
    S_F = np.array([
```

(continues on next page)

(continued from previous page)

```
        [0.0620, 0.0577],
        [0.0577, 0.0908]
    ])
    theta = np.array([0.0720, 0.0508, 0.0377, 0.0394, 0.0663, 0.0224, 0.0417, 0.0459])
    P       = np.linalg.cholesky(S_F)
    G_factor = B @ P

    k = G_factor.shape[1]
    gammas = [0.24, 0.28, 0.32, 0.36, 0.4, 0.44, 0.48]

    inf = 0.0 # This value has no significance

    # Variable offsets
    numvar = n
    voff_x = 0

    # Constraints offsets
    numcon = 1
    coff_bud = 0

    with mosek.Task() as task:
        task.set_Stream(mosek.streamtype.log, sys.stdout.write)

        # Holding variable x of length n
        # No other auxiliary variables are needed in this formulation
        task.appendvars(numvar)

        # Optionally we can give the variables names
        for j in range(0, n):
            task.putvarname(voff_x + j, "x[%d]" % (1 + j))

        # No short-selling in this model, all of x >= 0
        task.putvarboundsliceconst(voff_x, n, mosek.boundkey.lo, 0.0, inf)

        # One linear constraint: total budget
        task.appendcons(1)
        task.putconname(coff_bud, "budget")
        task.putaijlist([coff_bud] * n, range(voff_x, voff_x + n), [1.0] * n)
↪# e^T x
        rtemp = w + sum(x0)
        task.putconbound(coff_bud, mosek.boundkey.fx, rtemp, rtemp)     # equals w +␣
↪sum(x0)

        # Input (gamma, G_factor_T x, diag(sqrt(theta))*x) in the AFE (affine␣
↪expression) storage
        # We need k+n+1 rows and we fill them in in three parts
        task.appendafes(k + n + 1)
        # 1. The first affine expression = gamma, will be specified later
        # 2. The next k expressions comprise G_factor_T*x, we add them row by row
        #    transposing the matrix G_factor on the fly
        for i in range(0, k):
            task.putafefrow(i + 1, range(voff_x, voff_x + n), np.array(G_factor[:,i]))
        # 3. The remaining n rows contain sqrt(theta) on the diagonal
        task.putafefentrylist(range(k + 1, k + 1 + n), range(voff_x, voff_x + n), np.
↪sqrt(theta))
```

(continues on next page)

(continued from previous page)

```
        # Input the affine conic constraint (gamma, G_factor_T x,␣
↪diag(sqrt(theta))*x) \in QCone
        # Add the quadratic domain of dimension k+n+1
        qdom = task.appendquadraticconedomain(k + n + 1)
        # Add the constraint
        task.appendaccseq(qdom, 0, None)
        task.putaccname(0, "risk")

        # Objective: maximize expected return mu^T x
        task.putclist(range(voff_x, voff_x + n), mu)
        task.putobjsense(mosek.objsense.maximize)

        for gamma in gammas:
            # Specify gamma in ACC
            task.putafeg(0, gamma)

            # Dump the problem to a human readable PTF file.
            task.writedata("dump.ptf")

            # Solve the problem
            task.optimize()

            # Display solution summary for quick inspection of results.
            # In this simplified example we skip checks for problem and solution␣
↪status
            task.solutionsummary(mosek.streamtype.msg)

            # Check if the interior point solution is an optimal point
            solsta = task.getsolsta(mosek.soltype.itr)
            if (solsta != mosek.solsta.optimal):
                # See https://docs.mosek.com/latest/pythonapi/accessing-solution.html␣
↪about handling solution statuses.
                raise Exception(f"Unexpected solution status: {solsta}")

            # Retrieve results
            xx     = task.getxxslice(mosek.soltype.itr, voff_x, voff_x + n)
            expret = task.getprimalobj(mosek.soltype.itr)

            print(f'Expected return: {expret:.10e} Std. deviation: {gamma:.4e}')
            np.set_printoptions(precision=4)
            print(f'Optimal portfolio: {np.array(xx)}')
```

11.1.4 Slippage Cost

The basic Markowitz model assumes that there are no costs associated with trading the assets and that the returns of the assets are independent of the amount traded. Neither of those assumptions is usually valid in practice. Therefore, a more realistic model is

$$\begin{array}{lrcl} \text{maximize} & \mu^T x & & \\ \text{subject to} & e^T x + \sum_{j=1}^n T_j(\Delta x_j) & = & w + e^T x^0, \\ & x^T \Sigma x & \leq & \gamma^2, \\ & x & \geq & 0. \end{array} \tag{11.6}$$

Here Δx_j is the change in the holding of asset j i.e.

$$\Delta x_j = x_j - x_j^0$$

and $T_j(\Delta x_j)$ specifies the transaction costs when the holding of asset j is changed from its initial value. In the next two sections we show two different variants of this problem with two nonlinear cost functions T.

11.1.5 Market Impact Costs

If the initial wealth is fairly small and no short selling is allowed, then the holdings will be small and the traded amount of each asset must also be small. Therefore, it is reasonable to assume that the prices of the assets are independent of the amount traded. However, if a large volume of an asset is sold or purchased, the price, and hence return, can be expected to change. This effect is called market impact costs. It is common to assume that the market impact cost for asset j can be modeled by

$$T_j(\Delta x_j) = m_j|\Delta x_j|^{3/2}$$

where m_j is a constant that is estimated in some way by the trader. See [GK00] [p. 452] for details. From the Modeling Cookbook we know that $t \geq |z|^{3/2}$ can be modeled directly using the power cone $\mathcal{P}_3^{2/3,1/3}$:

$$\{(t,z) : t \geq |z|^{3/2}\} = \{(t,z) : (t,1,z) \in \mathcal{P}_3^{2/3,1/3}\}$$

Hence, it follows that $\sum_{j=1}^n T_j(\Delta x_j) = \sum_{j=1}^n m_j|x_j - x_j^0|^{3/2}$ can be modeled by $\sum_{j=1}^n m_j t_j$ under the constraints

$$\begin{array}{rl} z_j & = |x_j - x_j^0|, \\ (t_j, 1, z_j) & \in \mathcal{P}_3^{2/3,1/3}. \end{array}$$

Unfortunately this set of constraints is nonconvex due to the constraint

$$z_j = |x_j - x_j^0| \tag{11.7}$$

but in many cases the constraint may be replaced by the relaxed constraint

$$z_j \geq |x_j - x_j^0|, \tag{11.8}$$

For instance if the universe of assets contains a risk free asset then

$$z_j > |x_j - x_j^0| \tag{11.9}$$

cannot hold for an optimal solution.

If the optimal solution has the property (11.9) then the market impact cost within the model is larger than the true market impact cost and hence money are essentially considered garbage and removed by generating transaction costs. This may happen if a portfolio with very small risk is requested because the only way to obtain a small risk is to get rid of some of the assets by generating transaction costs. We generally assume that this is not the case and hence the models (11.7) and (11.8) are equivalent.

The above observations lead to

$$\begin{array}{lrcll} \text{maximize} & \mu^T x & & & \\ \text{subject to} & e^T x + m^T t & = & w + e^T x^0, & \\ & (\gamma, G^T x) & \in & \mathcal{Q}^{k+1}, & \\ & (t_j, 1, x_j - x_j^0) & \in & \mathcal{P}_3^{2/3,1/3}, & j = 1, \ldots, n, \\ & x & \geq & 0. & \end{array} \tag{11.10}$$

The revised budget constraint

$$e^T x + m^T t = w + e^T x^0$$

specifies that the initial wealth covers the investment and the transaction costs. It should be mentioned that transaction costs of the form

$$t_j \geq |z_j|^p$$

where $p > 1$ is a real number can be modeled with the power cone as

$$(t_j, 1, z_j) \in \mathcal{P}_3^{1/p,1-1/p}.$$

See the Modeling Cookbook for details.

Example code

Listing 11.5 demonstrates how to compute an optimal portfolio when market impact cost are included.

Listing 11.5: Implementation of model (11.10).

```
    with mosek.Task() as task:
        task.set_Stream(mosek.streamtype.log, sys.stdout.write)

        # Variables (vector of x, c, z)
        task.appendvars(numvar)
        for j in range(0, n):
            task.putvarname(voff_x + j, "x[%d]" % (j + 1))
            task.putvarname(voff_c + j, "c[%d]" % (j + 1))
            task.putvarname(voff_z + j, "z[%d]" % (j + 1))

        # Apply variable bounds (x >= 0, c and z free)
        task.putvarboundsliceconst(voff_x, voff_x + n, mosek.boundkey.lo, 0.0, inf)
        task.putvarboundsliceconst(voff_c, voff_c + n, mosek.boundkey.fr, -inf, inf)
        task.putvarboundsliceconst(voff_z, voff_z + n, mosek.boundkey.fr, -inf, inf)

        # Linear constraints
        # - Budget
        task.appendcons(1)
        task.putconname(coff_bud, "budget")
        task.putaijlist([coff_bud] * n, range(voff_x, voff_x + n), [1.0] * n)     # e^
↪T x
        task.putaijlist([coff_bud] * n, range(voff_c, voff_c + n), m)             # m^
↪T c
        rtemp = w + sum(x0)
        task.putconbound(coff_bud, mosek.boundkey.fx, rtemp, rtemp)     # equals w +␣
↪sum(x0)

        # - Absolute value
        task.appendcons(2 * n)
        for i in range(0, n):
            task.putconname(coff_abs1 + i, "zabs1[%d]" % (1 + i))
            task.putconname(coff_abs2 + i, "zabs2[%d]" % (1 + i))
        task.putaijlist(range(coff_abs1, coff_abs1 + n), range(voff_x, voff_x + n), n␣
↪* [-1.0])
        task.putaijlist(range(coff_abs1, coff_abs1 + n), range(voff_z, voff_z + n), n␣
↪* [1.0])
        task.putconboundlist(range(coff_abs1, coff_abs1 + n), [mosek.boundkey.lo] * n,
↪ [-x0[j] for j in range(0, n)], [inf] * n)
        task.putaijlist(range(coff_abs2, coff_abs2 + n), range(voff_x, voff_x + n), n␣
↪* [1.0])
        task.putaijlist(range(coff_abs2, coff_abs2 + n), range(voff_z, voff_z + n), n␣
↪* [1.0])
        task.putconboundlist(range(coff_abs2, coff_abs2 + n), [mosek.boundkey.lo] * n,
↪ x0, [inf] * n)

        # ACCs
        aoff_q = 0
        aoff_pow = k + 1
        # - (gamma, GTx) in Q(k+1)
        # The part of F and g for variable x:
        #     [0,  0, 0]      [gamma]
        # F = [GT, 0, 0], g = [0    ]
```

(continues on next page)

(continued from previous page)

```
        task.appendafes(k + 1)
        task.putafeg(aoff_q, gamma)
        for i in range(0, k):
            task.putafefrow(aoff_q + 1 + i, range(voff_x, voff_x + n), GT[i])
        qdom = task.appendquadraticconedomain(k + 1)
        task.appendaccseq(qdom, aoff_q, None)
        task.putaccname(aoff_q, "risk")

        # - (c_j, 1, z_j) in P3(2/3, 1/3)
        # The part of F and g for variables [c, z]:
        #     [0, I, 0]      [0]
        # F = [0, 0, I], g = [0]
        #     [0, 0, 0]      [1]
        task.appendafes(2 * n + 1)
        task.putafefentrylist(range(aoff_pow, aoff_pow + n), range(voff_c, voff_c +␣
↪n), [1.0] * n)
        task.putafefentrylist(range(aoff_pow + n, aoff_pow + 2 * n), range(voff_z,␣
↪voff_z + n), [1.0] * n)
        task.putafeg(aoff_pow + 2 * n, 1.0)
        # We use one row from F and g for both c_j and z_j, and the last row of F and␣
↪g for the constant 1.
        # NOTE: Here we reuse the last AFE and the power cone n times, but we store␣
↪them only once.
        powdom = task.appendprimalpowerconedomain(3, [2, 1])
        afe_list = [(aoff_pow + i, aoff_pow + 2 * n, aoff_pow + n + i) for i in␣
↪range(0, n)]
        flat_afe_list = [idx for sublist in afe_list for idx in sublist]
        task.appendaccs([powdom] * n, flat_afe_list, None)
        for i in range(0, n):
            task.putaccname(i + 1, "market_impact[%d]" % i)

        # Objective
        task.putclist(range(voff_x, voff_x + n), mu)
        task.putobjsense(mosek.objsense.maximize)

        # Turn all log output off.
        # task.putintparam(mosek.iparam.log,0)

        # Dump the problem to a human readable PTF file.
        task.writedata("dump.ptf")

        task.optimize()

        # Display the solution summary for quick inspection of results.
        task.solutionsummary(mosek.streamtype.msg)

        # Check if the interior point solution is an optimal point
        solsta = task.getsolsta(mosek.soltype.itr)
        if (solsta != mosek.solsta.optimal):
            # See https://docs.mosek.com/latest/pythonapi/accessing-solution.html␣
↪about handling solution statuses.
            raise Exception(f"Unexpected solution status: {solsta}")

        expret = 0.0
        x = task.getxxslice(mosek.soltype.itr, voff_x, voff_x + n)
        for j in range(0, n):
```

(continues on next page)

(continued from previous page)

```
            expret += mu[j] * x[j]

        print("\nExpected return %e for gamma %e\n" % (expret, gamma))
```

Note that in the following part of the code:

```
        task.putafeg(aoff_pow + 2 * n, 1.0)
        # We use one row from F and g for both c_j and z_j, and the last row of F and␣
↪g for the constant 1.
        # NOTE: Here we reuse the last AFE and the power cone n times, but we store␣
↪them only once.
        powdom = task.appendprimalpowerconedomain(3, [2, 1])
        afe_list = [(aoff_pow + i, aoff_pow + 2 * n, aoff_pow + n + i) for i in␣
↪range(0, n)]
        flat_afe_list = [idx for sublist in afe_list for idx in sublist]
        task.appendaccs([powdom] * n, flat_afe_list, None)
        for i in range(0, n):
            task.putaccname(i + 1, "market_impact[%d]" % i)
```

we create a sequence of power cones of the form $(t_k, 1, x_k - x_k^0) \in \mathcal{P}_3^{2/3,1/3}$. The power cones are determined by the sequence of exponents $(2, 1)$; we create a single domain to account for that.

Moreover, note that the second coordinate of all these affine conic constraints is the same affine expression equal to 1, and we use the feature that allows us to define this affine expression only once (as AFE number `aoff_pow + 2 * n`) and reuse it in all the ACCs.

11.1.6 Transaction Costs

Now assume there is a cost associated with trading asset j given by

$$T_j(\Delta x_j) = \begin{cases} 0, & \Delta x_j = 0, \\ f_j + g_j |\Delta x_j|, & \text{otherwise.} \end{cases}$$

Hence, whenever asset j is traded we pay a fixed setup cost f_j and a variable cost of g_j per unit traded. Given the assumptions about transaction costs in this section problem (11.6) may be formulated as

$$\begin{array}{lrcll} \text{maximize} & \mu^T x & & & \\ \text{subject to} & e^T x + f^T y + g^T z & = & w + e^T x^0, & \\ & (\gamma, G^T x) & \in & \mathcal{Q}^{k+1}, & \\ & z_j & \geq & x_j - x_j^0, & j = 1, \ldots, n, \\ & z_j & \geq & x_j^0 - x_j, & j = 1, \ldots, n, \\ & z_j & \leq & U_j y_j, & j = 1, \ldots, n, \\ & y_j & \in & \{0, 1\}, & j = 1, \ldots, n, \\ & x & \geq & 0. & \end{array} \tag{11.11}$$

First observe that

$$z_j \geq |x_j - x_j^0| = |\Delta x_j|.$$

We choose U_j as some a priori upper bound on the amount of trading in asset j and therefore if $z_j > 0$ then $y_j = 1$ has to be the case. This implies that the transaction cost for asset j is given by

$$f_j y_j + g_j z_j.$$

Example code

The following example code demonstrates how to compute an optimal portfolio when transaction costs are included.

Listing 11.6: Code solving problem (11.11).

```
    with mosek.Task() as task:
        task.set_Stream(mosek.streamtype.log, sys.stdout.write)

        # Variables (vector of x, z, y)
        task.appendvars(numvar)
        for j in range(0, n):
            task.putvarname(voff_x + j, "x[%d]" % (j + 1))
            task.putvarname(voff_z + j, "z[%d]" % (j + 1))
            task.putvarname(voff_y + j, "y[%d]" % (j + 1))

        # Apply variable bounds (x >= 0, z free, y binary)
        task.putvarboundsliceconst(voff_x, voff_x + n, mosek.boundkey.lo, 0.0, inf)
        task.putvarboundsliceconst(voff_z, voff_z + n, mosek.boundkey.fr, -inf, inf)
        task.putvarboundsliceconst(voff_y, voff_y + n, mosek.boundkey.ra, 0.0, 1.0)
        task.putvartypelist(range(voff_y, voff_y + n), [mosek.variabletype.type_int]␣
↪* n)

        # Linear constraints
        # - Budget
        task.appendcons(1)
        task.putconname(coff_bud, "budget")
        task.putaijlist([0] * n, range(voff_x, voff_x + n), [1.0] * n)     # e^T x
        task.putaijlist([0] * n, range(voff_z, voff_z + n), g)             # g^T z
        task.putaijlist([0] * n, range(voff_y, voff_y + n), f)             # f^T y
        U = w + sum(x0)
        task.putconbound(coff_bud, mosek.boundkey.fx, U, U)     # equals w + sum(x0)

        # - Absolute value
        task.appendcons(2 * n)
        for i in range(0, n):
            task.putconname(coff_abs1 + i, "zabs1[%d]" % (1 + i))
            task.putconname(coff_abs2 + i, "zabs2[%d]" % (1 + i))
        task.putaijlist(range(coff_abs1, coff_abs1 + n), range(voff_x, voff_x + n), [-
↪1.0] * n)
        task.putaijlist(range(coff_abs1, coff_abs1 + n), range(voff_z, voff_z + n),␣
↪[1.0] * n)
        task.putconboundlist(range(coff_abs1, coff_abs1 + n), [mosek.boundkey.lo] * n,
↪ [-x0[j] for j in range(0, n)], [inf] * n)
        task.putaijlist(range(coff_abs2, coff_abs2 + n), range(voff_x, voff_x + n),␣
↪[1.0] * n)
        task.putaijlist(range(coff_abs2, coff_abs2 + n), range(voff_z, voff_z + n),␣
↪[1.0] * n)
        task.putconboundlist(range(coff_abs2, coff_abs2 + n), [mosek.boundkey.lo] * n,
↪ x0, [inf] * n)

        # - Switch
        task.appendcons(n)
        for i in range(0, n):
            task.putconname(coff_swi + i, "switch[%d]" % (1 + i))
        task.putaijlist(range(coff_swi, coff_swi + n), range(voff_z, voff_z + n), [1.
↪0] * n)
```

(continues on next page)

(continued from previous page)

```
        task.putaijlist(range(coff_swi, coff_swi + n), range(voff_y, voff_y + n), [-
↪U] * n)
        task.putconboundlist(range(coff_swi, coff_swi + n), [mosek.boundkey.up] * n,␣
↪[-inf] * n, [0.0] * n)

        # ACCs
        aoff_q = 0
        # - (gamma, GTx) in Q(k+1)
        # The part of F and g for variable x:
        #     [0,  0, 0]      [gamma]
        # F = [GT, 0, 0], g = [0    ]
        task.appendafes(k + 1)
        task.putafeg(aoff_q, gamma)
        for i in range(0, k):
            task.putafefrow(aoff_q + i + 1, range(voff_x, voff_x + n), GT[i])
        qdom = task.appendquadraticconedomain(k + 1)
        task.appendaccseq(qdom, aoff_q, None)
        task.putaccname(0, "risk")

        # Objective
        task.putclist(range(voff_x, voff_x + n), mu)
        task.putobjsense(mosek.objsense.maximize)

        # Turn all log output off.
        # task.putintparam(mosek.iparam.log,0)

        # Dump the problem to a human readable PTF file.
        task.writedata("dump.ptf")

        task.optimize()

        # Display the solution summary for quick inspection of results.
        task.solutionsummary(mosek.streamtype.msg)

        # Check if the integer solution is an optimal point
        solsta = task.getsolsta(mosek.soltype.itg)
        if (solsta != mosek.solsta.integer_optimal):
            # See https://docs.mosek.com/latest/pythonapi/accessing-solution.html␣
↪about handling solution statuses.
            raise Exception(f"Unexpected solution status: {solsta}")

        expret = 0.0
        x = task.getxxslice(mosek.soltype.itg, voff_x, voff_x + n)
        for j in range(0, n):
            expret += mu[j] * x[j]

        tcost = 0.0
        z = task.getxxslice(mosek.soltype.itg, voff_z, voff_z + n)
        y = task.getxxslice(mosek.soltype.itg, voff_y, voff_y + n)
        for j in range(0, n):
            tcost += g[j] * z[j] + f[j] * y[j]

        print("\nExpected return %e for gamma %e. Transaction cost: %e\n" % (expret,␣
↪gamma, tcost))
```

11.1.7 Cardinality constraints

Another method to reduce costs involved with processing transactions is to only change positions in a small number of assets. In other words, at most K of the differences $|\Delta x_j| = |x_j - x_j^0|$ are allowed to be non-zero, where K is (much) smaller than the total number of assets n.

This type of constraint can be again modeled by introducing a binary variable y_j which indicates if $\Delta x_j \neq 0$ and bounding the sum of y_j. The basic Markowitz model then gets updated as follows:

$$\begin{array}{lrcll} \text{maximize} & \mu^T x & & & \\ \text{subject to} & e^T x & = & w + e^T x^0, & \\ & (\gamma, G^T x) & \in & \mathcal{Q}^{k+1}, & \\ & z_j & \geq & x_j - x_j^0, & j = 1, \ldots, n, \\ & z_j & \geq & x_j^0 - x_j, & j = 1, \ldots, n, \\ & z_j & \leq & U_j y_j, & j = 1, \ldots, n, \\ & y_j & \in & \{0, 1\}, & j = 1, \ldots, n, \\ & e^T y & \leq & K, & \\ & x & \geq & 0, & \end{array} \tag{11.12}$$

were U_j is some a priori chosen upper bound on the amount of trading in asset j.

Example code

The following example code demonstrates how to compute an optimal portfolio with cardinality bounds.

Listing 11.7: Code solving problem (11.12).

```
def markowitz_with_card(n, k, x0, w, gamma, mu, GT, K):
    with mosek.Task() as task:
        task.set_Stream(mosek.streamtype.log, sys.stdout.write)

        # Offset of variables.
        numvar = 3 * n
        voff_x, voff_z, voff_y = 0, n, 2 * n

        # Offset of constraints.
        numcon = 3 * n + 2
        coff_bud, coff_abs1, coff_abs2, coff_swi, coff_card = 0, 1, 1 + n, 1 + 2 * n,␣
↪1 + 3 * n

        # Variables (vector of x, z, y)
        task.appendvars(numvar)
        for j in range(0, n):
            task.putvarname(voff_x + j, "x[%d]" % (j + 1))
            task.putvarname(voff_z + j, "z[%d]" % (j + 1))
            task.putvarname(voff_y + j, "y[%d]" % (j + 1))

        # Apply variable bounds (x >= 0, z free, y binary)
        task.putvarboundsliceconst(voff_x, voff_x + n, mosek.boundkey.lo, 0.0, inf)
        task.putvarboundsliceconst(voff_z, voff_z + n, mosek.boundkey.fr, -inf, inf)
        task.putvarboundsliceconst(voff_y, voff_y + n, mosek.boundkey.ra, 0.0, 1.0)
        task.putvartypelist(range(voff_y, voff_y + n), [mosek.variabletype.type_int]␣
↪* n)

        # Linear constraints
        # - Budget
        task.appendcons(1)
        task.putconname(coff_bud, "budget")
```

(continues on next page)

(continued from previous page)

```
        task.putaijlist([coff_bud] * n, range(voff_x, voff_x + n), [1.0] * n)     # e^
↪T x
        U = w + sum(x0)
        task.putconbound(coff_bud, mosek.boundkey.fx, U, U)     # = w + sum(x0)

        # - Absolute value
        task.appendcons(2 * n)
        for i in range(0, n):
            task.putconname(coff_abs1 + i, "zabs1[%d]" % (1 + i))
            task.putconname(coff_abs2 + i, "zabs2[%d]" % (1 + i))
        task.putaijlist(range(coff_abs1, coff_abs1 + n), range(voff_x, voff_x + n), [-
↪1.0] * n)
        task.putaijlist(range(coff_abs1, coff_abs1 + n), range(voff_z, voff_z + n),␣
↪[1.0] * n)
        task.putconboundlist(range(coff_abs1, coff_abs1 + n), [mosek.boundkey.lo] * n,
↪ [-x0[j] for j in range(0, n)], [inf] * n)
        task.putaijlist(range(coff_abs2, coff_abs2 + n), range(voff_x, voff_x + n),␣
↪[1.0] * n)
        task.putaijlist(range(coff_abs2, coff_abs2 + n), range(voff_z, voff_z + n),␣
↪[1.0] * n)
        task.putconboundlist(range(coff_abs2, coff_abs2 + n), [mosek.boundkey.lo] * n,
↪ x0, [inf] * n)

        # - Switch
        task.appendcons(n)
        for i in range(0, n):
            task.putconname(coff_swi + i, "switch[%d]" % (1 + i))
        task.putaijlist(range(coff_swi, coff_swi + n), range(voff_z, voff_z + n), [1.
↪0] * n)
        task.putaijlist(range(coff_swi, coff_swi + n), range(voff_y, voff_y + n), [-
↪U] * n)
        task.putconboundlist(range(coff_swi, coff_swi + n), [mosek.boundkey.up] * n,␣
↪[-inf] * n, [0.0] * n)

        # - Cardinality
        task.appendcons(1)
        task.putconname(coff_card, "cardinality")
        task.putaijlist([coff_card] * n, range(voff_y, voff_y + n), [1.0] * n)     # e^
↪T y
        task.putconbound(coff_card, mosek.boundkey.up, -inf, K)            # <= K

        # ACCs
        aoff_q = 0
        # - (gamma, GTx) in Q(k+1)
        # The part of F and g for variable x:
        #     [0,  0, 0]      [gamma]
        # F = [GT, 0, 0], g = [0    ]
        task.appendafes(k + 1)
        task.putafeg(aoff_q, gamma)
        for i in range(0, k):
            task.putafefrow(aoff_q + i + 1, range(voff_x, voff_x + n), GT[i])
        qdom = task.appendquadraticconedomain(k + 1)
        task.appendaccseq(qdom, aoff_q, None)
        task.putaccname(0, "risk")

        # Objective
```

(continues on next page)

(continued from previous page)

```
        task.putclist(range(voff_x, voff_x + n), mu)
        task.putobjsense(mosek.objsense.maximize)

        # Turn all log output off.
        task.putintparam(mosek.iparam.log,0)

        # Dump the problem to a human readable PTF file.
        task.writedata("dump.ptf")

        task.optimize()

        # Check if the integer solution is an optimal point
        solsta = task.getsolsta(mosek.soltype.itg)
        if (solsta != mosek.solsta.integer_optimal):
            # See https://docs.mosek.com/latest/pythonapi/accessing-solution.html␣
↪about handling solution statuses.
            raise Exception(f"Unexpected solution status: {solsta}")

        # Display the solution summary for quick inspection of results.
        #task.solutionsummary(mosek.streamtype.msg)

        return task.getxxslice(mosek.soltype.itg, voff_x + 0, voff_x + n)
```

If we solve our running example with $K = 1, \ldots, n$ then we get the following solutions, with increasing expected returns:

```
Bound 1   Solution: 0.0000e+00   0.0000e+00   1.0000e+00   0.0000e+00   0.0000e+00  ␣
↪0.0000e+00   0.0000e+00   0.0000e+00
Bound 2   Solution: 0.0000e+00   0.0000e+00   3.5691e-01   0.0000e+00   0.0000e+00  ␣
↪6.4309e-01   -0.0000e+00  0.0000e+00
Bound 3   Solution: 0.0000e+00   0.0000e+00   1.9258e-01   0.0000e+00   0.0000e+00  ␣
↪5.4592e-01   2.6150e-01   0.0000e+00
Bound 4   Solution: 0.0000e+00   0.0000e+00   2.0391e-01   0.0000e+00   6.7098e-02  ␣
↪4.9181e-01   2.3718e-01   0.0000e+00
Bound 5   Solution: 0.0000e+00   3.1970e-02   1.7028e-01   0.0000e+00   7.0741e-02  ␣
↪4.9551e-01   2.3150e-01   0.0000e+00
Bound 6   Solution: 0.0000e+00   3.1970e-02   1.7028e-01   0.0000e+00   7.0740e-02  ␣
↪4.9551e-01   2.3150e-01   0.0000e+00
Bound 7   Solution: 0.0000e+00   3.1970e-02   1.7028e-01   0.0000e+00   7.0740e-02  ␣
↪4.9551e-01   2.3150e-01   0.0000e+00
Bound 8   Solution: 1.9557e-10   2.6992e-02   1.6706e-01   2.9676e-10   7.1245e-02  ␣
↪4.9559e-01   2.2943e-01   9.6905e-03
```

11.2 Logistic regression

Logistic regression is an example of a binary classifier, where the output takes one two values 0 or 1 for each data point. We call the two values *classes*.

Formulation as an optimization problem

Define the sigmoid function

$$S(x) = \frac{1}{1+\exp(-x)}.$$

Next, given an observation $x \in \mathbb{R}^d$ and a weights $\theta \in \mathbb{R}^d$ we set

$$h_\theta(x) = S(\theta^T x) = \frac{1}{1+\exp(-\theta^T x)}.$$

The weights vector θ is part of the setup of the classifier. The expression $h_\theta(x)$ is interpreted as the probability that x belongs to class 1. When asked to classify x the returned answer is

$$x \mapsto \begin{cases} 1 & h_\theta(x) \geq 1/2, \\ 0 & h_\theta(x) < 1/2. \end{cases}$$

When training a logistic regression algorithm we are given a sequence of training examples x_i, each labelled with its class $y_i \in \{0,1\}$ and we seek to find the weights θ which maximize the likelihood function

$$\prod_i h_\theta(x_i)^{y_i}(1-h_\theta(x_i))^{1-y_i}.$$

Of course every single y_i equals 0 or 1, so just one factor appears in the product for each training data point. By taking logarithms we can define the logistic loss function:

$$J(\theta) = -\sum_{i:y_i=1} \log(h_\theta(x_i)) - \sum_{i:y_i=0} \log(1-h_\theta(x_i)).$$

The training problem with regularization (a standard technique to prevent overfitting) is now equivalent to

$$\min_\theta J(\theta) + \lambda\|\theta\|_2.$$

This can equivalently be phrased as

$$\begin{array}{lrlll} \text{minimize} & \sum_i t_i + \lambda r & & & \\ \text{subject to} & t_i & \geq -\log(h_\theta(x)) & = \log(1+\exp(-\theta^T x_i)) & \text{if } y_i = 1, \\ & t_i & \geq -\log(1-h_\theta(x)) & = \log(1+\exp(\theta^T x_i)) & \text{if } y_i = 0, \\ & r & \geq \|\theta\|_2. & & \end{array} \tag{11.13}$$

Implementation

As can be seen from (11.13) the key point is to implement the softplus bound $t \geq \log(1+e^u)$, which is the simplest example of a log-sum-exp constraint for two terms. Here t is a scalar variable and u will be the affine expression of the form $\pm\theta^T x_i$. This is equivalent to

$$\exp(u-t) + \exp(-t) \leq 1$$

and further to

$$\begin{array}{rcll} (z_1, 1, u-t) & \in & K_{\exp} & (z_1 \geq \exp(u-t)), \\ (z_2, 1, -t) & \in & K_{\exp} & (z_2 \geq \exp(-t)), \\ z_1 + z_2 & \leq & 1. & \end{array} \tag{11.14}$$

This formulation can be entered using affine conic constraints (see Sec. 6.2).

Listing 11.8: Implementation of $t \geq \log(1 + e^u)$ as in (11.14).

```
# Adds ACCs for t_i >= log ( 1 + exp((1-2*y[i]) * theta' * X[i]) )
# Adds auxiliary variables, AFE rows and constraints
def softplus(task, d, n, theta, t, X, y):
    nvar = task.getnumvar()
    ncon = task.getnumcon()
    nafe = task.getnumafe()
    task.appendvars(2*n)     # z1, z2
    task.appendcons(n)       # z1 + z2 = 1
    task.appendafes(4*n)     #theta * X[i] - t[i], -t[i], z1[i], z2[i]
    z1, z2 = nvar, nvar+n
    zcon = ncon
    thetaafe, tafe, z1afe, z2afe = nafe, nafe+n, nafe+2*n, nafe+3*n
    for i in range(n):
        task.putvarname(z1+i,f"z1[{i}]")
        task.putvarname(z2+i,f"z2[{i}]")

    # z1 + z2 = 1
    task.putaijlist(range(zcon, zcon+n), range(z1, z1+n), [1]*n)
    task.putaijlist(range(zcon, zcon+n), range(z2, z2+n), [1]*n)
    task.putconboundsliceconst(zcon, zcon+n, boundkey.fx, 1, 1)
    task.putvarboundsliceconst(nvar, nvar+2*n, boundkey.fr, -inf, inf)

    # Affine conic expressions
    afeidx, varidx, fval = [], [], []

    ## Thetas
    for i in range(n):
      for j in range(d):
        afeidx.append(thetaafe + i)
        varidx.append(theta + j)
        fval.append(-X[i][j] if y[i]==1 else X[i][j])

    # -t[i]
    afeidx.extend([thetaafe + i for i in range(n)] + [tafe + i for i in range(n)])
    varidx.extend([t + i for i in range(n)] + [t + i for i in range(n)])
    fval.extend([-1.0]*(2*n))

    # z1, z2
    afeidx.extend([z1afe + i for i in range(n)] + [z2afe + i for i in range(n)])
    varidx.extend([z1 + i for i in range(n)] + [z2 + i for i in range(n)])
    fval.extend([1.0]*(2*n))

    # Add the expressions
    task.putafefentrylist(afeidx, varidx, fval)

    # Add a single row with the constant expression "1.0"
    oneafe = task.getnumafe()
    task.appendafes(1)
    task.putafeg(oneafe, 1.0)

    # Add an exponential cone domain
    expdomain = task.appendprimalexpconedomain()

    # Conic constraints
    acci = task.getnumacc()
```

(continues on next page)

(continued from previous page)

```
    for i in range(n):
      task.appendacc(expdomain, [z1afe+i, oneafe, thetaafe+i], None)
      task.appendacc(expdomain, [z2afe+i, oneafe, tafe+i], None)
      task.putaccname(acci,  f"z1:theta[{i}]")
      task.putaccname(acci+1,f"z2:t[{i}]")
      acci += 2
```

Once we have this subroutine, it is easy to implement a function that builds the regularized loss function model (11.13).

Listing 11.9: Implementation of (11.13).

```
# Model logistic regression (regularized with full 2-norm of theta)
# X - n x d matrix of data points
# y - length n vector classifying training points
# lamb - regularization parameter
def logisticRegression(X, y, lamb=1.0):
    n, d = int(X.shape[0]), int(X.shape[1])          # num samples, dimension

    with Task() as task:
        # Variables [r; theta; t; u]
        nvar = 1+d+2*n
        task.appendvars(nvar)
        task.putvarboundsliceconst(0, nvar, boundkey.fr, -inf, inf)
        r, theta, t = 0, 1, 1+d
        task.putvarname(r,"r");
        for j in range(d): task.putvarname(theta+j,f"theta[{j}]");
        for j in range(n): task.putvarname(t+j,f"t[{j}]");

        # Objective lambda*r + sum(t)
        task.putobjsense(objsense.minimize)
        task.putcj(r, lamb)
        task.putclist(range(t, t+n), [1.0]*n)

        # Softplus function constraints
        softplus(task, d, n, theta, t, X, y);

        # Regularization
        # Append a sequence of linear expressions (r, theta) to F
        numafe = task.getnumafe()
        task.appendafes(1+d)
        task.putafefentry(numafe, r, 1.0)
        for i in range(d):
          task.putafefentry(numafe + i + 1, theta + i, 1.0)

        # Add the constraint
        task.appendaccseq(task.appendquadraticconedomain(1+d), numafe, None)

        # Solution
        task.writedata('logistic.ptf')
        task.optimize()
        xx = task.getxxslice(soltype.itr, theta, theta+d)

        return xx
```

Example: 2D dataset fitting

In the next figure we apply logistic regression to the training set of 2D points taken from the example ex2data2.txt . The two-dimensional dataset was converted into a feature vector $x \in \mathbb{R}^{28}$ using monomial coordinates of degrees at most 6.

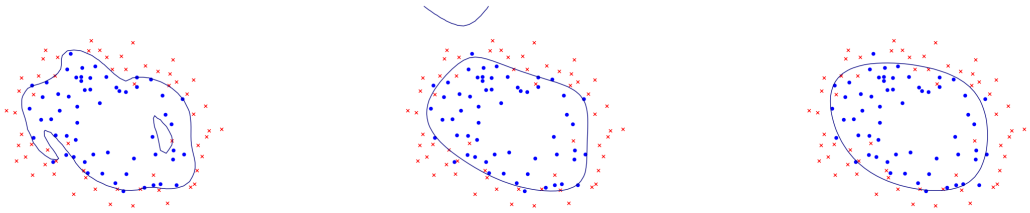

Fig. 11.2: Logistic regression example with none, medium and strong regularization (small, medium, large λ). Without regularization we get obvious overfitting.

11.3 Concurrent optimizer

The idea of the concurrent optimizer is to run multiple optimizations of **the same problem** simultaneously, and pick the one that provides the fastest or best answer. This approach is especially useful for problems which require a very long time and it is hard to say in advance which optimizer or algorithm will perform best.

The major applications of concurrent optimization we describe in this section are:

- Using the interior-point and simplex optimizers simultaneously on a linear problem. Note that any solution present in the task will also be used for hot-starting the simplex algorithms. One possible scenario would therefore be running a hot-start simplex in parallel with interior point, taking advantage of both the stability of the interior-point method and the ability of the simplex method to use an initial solution.
- Using multiple instances of the mixed-integer optimizer to solve many copies of one mixed-integer problem. This is not in contradiction with the run-to-run determinism of **MOSEK** if a different value of the MIO seed parameter *iparam.mio_seed* is set in each instance. As a result each setting leads to a different optimizer run (each of them being deterministic in its own right).

The downloadable file contains usage examples of both kinds.

11.3.1 Common setup

We first define a method that runs a number of optimization tasks in parallel, using the standard multithreading setup available in the language. All tasks register for a callback function which will signal them to interrupt as soon as the first task completes successfully (with response code *rescode.ok*).

Listing 11.10: Simple callback function which signals the optimizer to stop.

```
# Defines a Mosek callback function whose only function
# is to indicate if the optimizer should be stopped.
stop = False
firstStop = -1
def cbFun(code):
  return 1 if stop else 0
```

When all remaining tasks respond to the stop signal, response codes and statuses are returned to the caller, together with the index of the task which won the race.

Listing 11.11: A routine for parallel task race.

```
def runTask(num, task, res, trm):
  global stop
  global firstStop
  try:
    trm[num] = task.optimize();
    res[num] = mosek.rescode.ok
  except mosek.MosekException as e:
    trm[num] = mosek.rescode.err_unknown
    res[num] = e.errno
  finally:
    # If this finished with success, inform other tasks to interrupt
    if res[num] == mosek.rescode.ok:
      if not stop:
        firstStop = num
      stop = True

def optimize(tasks):
  n = len(tasks)
  res = [ mosek.rescode.err_unknown ] * n
  trm = [ mosek.rescode.err_unknown ] * n

  # Set a callback function
  for t in tasks:
    t.set_Progress(cbFun)

  # Start parallel optimizations, one per task
  jobs = [ Thread(target=runTask, args=(i, tasks[i], res, trm)) for i in range(n) ]
  for j in jobs:
    j.start()
  for j in jobs:
    j.join()

  # For debugging, print res and trm codes for all optimizers
  for i in range(n):
    print("Optimizer  {0}   res {1}   trm {2}".format(i, res[i], trm[i]))

  return firstStop, res, trm
```

11.3.2 Linear optimization

We use the multithreaded setup to run the interior-point and simplex optimizers simultaneously on a linear problem. The next methods simply clones the given task and sets a different optimizer for each. The result is the clone which finished first.

Listing 11.12: Concurrent optimization with different optimizers.

```
def optimizeconcurrent(task, optimizers):
  n = len(optimizers)
  tasks = [ mosek.Task(task) for _ in range(n) ]

  # Choose various optimizers for cloned tasks
  for i in range(n):
    tasks[i].putintparam(mosek.iparam.optimizer, optimizers[i])

  # Solve tasks in parallel
```

(continues on next page)

(continued from previous page)

```
  firstOK, res, trm = optimize(tasks)

  if firstOK >= 0:
    return firstOK, tasks[firstOK], trm[firstOK], res[firstOK]
  else:
    return -1, None, None, None
```

It remains to call the method with a choice of optimizers, for example:

Listing 11.13: Calling concurrent linear optimization.

```
        optimizers = [
          mosek.optimizertype.conic,
          mosek.optimizertype.dual_simplex,
          mosek.optimizertype.primal_simplex
        ]

        idx, t, trm, res = optimizeconcurrent(task, optimizers)
```

11.3.3 Mixed-integer optimization

We use the multithreaded setup to run many, differently seeded copies of the mixed-integer optimizer. This approach is most useful for hard problems where we don't expect an optimal solution in reasonable time. The input task would typically contain a time limit. It is possible that all the cloned tasks reach the time limit, in which case it doesn't really mater which one terminated first. Instead we examine all the task clones for the best objective value.

Listing 11.14: Concurrent optimization of a mixed-integer problem.

```
def optimizeconcurrentMIO(task, seeds):
  n = len(seeds)
  tasks = [ mosek.Task(task) for _ in range(n) ]

  # Choose various seeds for cloned tasks
  for i in range(n):
    tasks[i].putintparam(mosek.iparam.mio_seed, seeds[i])

  # Solve tasks in parallel
  firstOK, res, trm = optimize(tasks)

  if firstOK >= 0:
    # Pick the task that ended with res = ok
    # and contains an integer solution with best objective value
    sense = task.getobjsense();
    bestObj = 1.0e+10 if sense == mosek.objsense.minimize else -1.0e+10
    bestPos = -1

    for i in range(n):
      print("{0}   {1}".format(i,tasks[i].getprimalobj(mosek.soltype.itg)))

    for i in range(n):
      if ((res[i] == mosek.rescode.ok) and
          (tasks[i].getsolsta(mosek.soltype.itg) == mosek.solsta.prim_feas or
           tasks[i].getsolsta(mosek.soltype.itg) == mosek.solsta.integer_optimal) and
         ((tasks[i].getprimalobj(mosek.soltype.itg) < bestObj)
           if (sense == mosek.objsense.minimize) else
          (tasks[i].getprimalobj(mosek.soltype.itg) > bestObj))):
```

(continues on next page)

(continued from previous page)

```
        bestObj = tasks[i].getprimalobj(mosek.soltype.itg)
        bestPos = i

    if bestPos >= 0:
      return bestPos, tasks[bestPos], trm[bestPos], res[bestPos]

  return -1, None, None, None
```

It remains to call the method with a choice of seeds, for example:

Listing 11.15: Calling concurrent integer optimization.

```
        seeds = [ 42, 13, 71749373 ]

        idx, t, trm, res = optimizeconcurrentMIO(task, seeds)
```

Chapter 12

Problem Formulation and Solutions

In this chapter we will discuss the following topics:

- The formal, mathematical formulations of the problem types that **MOSEK** can solve and their duals.
- The solution information produced by **MOSEK**.
- The infeasibility certificate produced by **MOSEK** if the problem is infeasible.

For the underlying mathematical concepts, derivations and proofs see the Modeling Cookbook or any book on convex optimization. This chapter explains how the related data is organized specifically within the **MOSEK** API.

12.1 Linear Optimization

MOSEK accepts linear optimization problems of the form

$$\begin{array}{lccccc} \text{minimize} & & & c^T x + c^f & & \\ \text{subject to} & l^c & \leq & Ax & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x, \end{array} \tag{12.1}$$

where

- m is the number of constraints.
- n is the number of decision variables.
- $x \in \mathbb{R}^n$ is a vector of decision variables.
- $c \in \mathbb{R}^n$ is the linear part of the objective function.
- $c^f \in \mathbb{R}$ is a constant term in the objective
- $A \in \mathbb{R}^{m \times n}$ is the constraint matrix.
- $l^c \in \mathbb{R}^m$ is the lower limit on the activity for the constraints.
- $u^c \in \mathbb{R}^m$ is the upper limit on the activity for the constraints.
- $l^x \in \mathbb{R}^n$ is the lower limit on the activity for the variables.
- $u^x \in \mathbb{R}^n$ is the upper limit on the activity for the variables.

Lower and upper bounds can be infinite, or in other words the corresponding bound may be omitted.

A primal solution (x) is *(primal) feasible* if it satisfies all constraints in (12.1). If (12.1) has at least one primal feasible solution, then (12.1) is said to be (primal) feasible. In case (12.1) does not have a feasible solution, the problem is said to be *(primal) infeasible*

12.1.1 Duality for Linear Optimization

Corresponding to the primal problem (12.1), there is a dual problem

$$\begin{array}{lrcl} \text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x + c^f & & \\ & A^T y + s_l^x - s_u^x & = & c, \\ \text{subject to} & -y + s_l^c - s_u^c & = & 0, \\ & s_l^c, s_u^c, s_l^x, s_u^x & \geq & 0, \end{array} \tag{12.2}$$

where

- s_l^c are the dual variables for lower bounds of constraints,
- s_u^c are the dual variables for upper bounds of constraints,
- s_l^x are the dual variables for lower bounds of variables,
- s_u^x are the dual variables for upper bounds of variables.

If a bound in the primal problem is plus or minus infinity, the corresponding dual variable is fixed at 0, and we use the convention that the product of the bound value and the corresponding dual variable is 0. This is equivalent to removing the corresponding dual variable from the dual problem. For example:

$$l_j^x = -\infty \quad \Rightarrow \quad (s_l^x)_j = 0 \text{ and } l_j^x \cdot (s_l^x)_j = 0.$$

A solution

$$(y, s_l^c, s_u^c, s_l^x, s_u^x)$$

to the dual problem is feasible if it satisfies all the constraints in (12.2). If (12.2) has at least one feasible solution, then (12.2) is *(dual) feasible*, otherwise the problem is *(dual) infeasible*.

A solution

$$(x^*, y^*, (s_l^c)^*, (s_u^c)^*, (s_l^x)^*, (s_u^x)^*)$$

is denoted a *primal-dual feasible solution*, if (x^*) is a solution to the primal problem (12.1) and $(y^*, (s_l^c)^*, (s_u^c)^*, (s_l^x)^*, (s_u^x)^*)$ is a solution to the corresponding dual problem (12.2). We also define an auxiliary vector

$$(x^c)^* := Ax^*$$

containing the activities of linear constraints.

For a primal-dual feasible solution we define the *duality gap* as the difference between the primal and the dual objective value,

$$\begin{array}{l} c^T x^* + c^f - \left\{ (l^c)^T (s_l^c)^* - (u^c)^T (s_u^c)^* + (l^x)^T (s_l^x)^* - (u^x)^T (s_u^x)^* + c^f \right\} \\ = \sum_{i=0}^{m-1} \left[(s_l^c)_i^* ((x_i^c)^* - l_i^c) + (s_u^c)_i^* (u_i^c - (x_i^c)^*) \right] \\ + \sum_{j=0}^{n-1} \left[(s_l^x)_j^* (x_j - l_j^x) + (s_u^x)_j^* (u_j^x - x_j^*) \right] \geq 0 \end{array} \tag{12.3}$$

where the first relation can be obtained by transposing and multiplying the dual constraints (12.2) by x^* and $(x^c)^*$ respectively, and the second relation comes from the fact that each term in each sum is nonnegative. It follows that the primal objective will always be greater than or equal to the dual objective.

It is well-known that a linear optimization problem has an optimal solution if and only if there exist feasible primal-dual solution so that the duality gap is zero, or, equivalently, that the *complementarity conditions*

$$\begin{array}{rcll} (s_l^c)_i^* ((x_i^c)^* - l_i^c) & = & 0, & i = 0, \ldots, m-1, \\ (s_u^c)_i^* (u_i^c - (x_i^c)^*) & = & 0, & i = 0, \ldots, m-1, \\ (s_l^x)_j^* (x_j^* - l_j^x) & = & 0, & j = 0, \ldots, n-1, \\ (s_u^x)_j^* (u_j^x - x_j^*) & = & 0, & j = 0, \ldots, n-1, \end{array}$$

are satisfied.

If (12.1) has an optimal solution and **MOSEK** solves the problem successfully, both the primal and dual solution are reported, including a status indicating the exact state of the solution.

12.1.2 Infeasibility for Linear Optimization

Primal Infeasible Problems

If the problem (12.1) is infeasible (has no feasible solution), **MOSEK** will report a certificate of primal infeasibility: The dual solution reported is the certificate of infeasibility, and the primal solution is undefined.

A certificate of primal infeasibility is a feasible solution to the modified dual problem

$$\begin{array}{ll} \text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x \\ \text{subject to} & \\ & \begin{array}{c} A^T y + s_l^x - s_u^x = 0, \\ -y + s_l^c - s_u^c = 0, \\ s_l^c, s_u^c, s_l^x, s_u^x \geq 0, \end{array} \end{array} \tag{12.4}$$

such that the objective value is strictly positive, i.e. a solution

$$(y^*, (s_l^c)^*, (s_u^c)^*, (s_l^x)^*, (s_u^x)^*)$$

to (12.4) so that

$$(l^c)^T (s_l^c)^* - (u^c)^T (s_u^c)^* + (l^x)^T (s_l^x)^* - (u^x)^T (s_u^x)^* > 0.$$

Such a solution implies that (12.4) is unbounded, and that (12.1) is infeasible.

Dual Infeasible Problems

If the problem (12.2) is infeasible (has no feasible solution), **MOSEK** will report a certificate of dual infeasibility: The primal solution reported is the certificate of infeasibility, and the dual solution is undefined.

A certificate of dual infeasibility is a feasible solution to the modified primal problem

$$\begin{array}{lccccc} \text{minimize} & & & c^T x & & \\ \text{subject to} & \hat{l}^c & \leq & Ax & \leq & \hat{u}^c, \\ & \hat{l}^x & \leq & x & \leq & \hat{u}^x, \end{array} \tag{12.5}$$

where

$$\hat{l}_i^c = \left\{ \begin{array}{ll} 0 & \text{if } l_i^c > -\infty, \\ -\infty & \text{otherwise,} \end{array} \right\} \quad \text{and} \quad \hat{u}_i^c := \left\{ \begin{array}{ll} 0 & \text{if } u_i^c < \infty, \\ \infty & \text{otherwise,} \end{array} \right\}$$

and

$$\hat{l}_j^x = \left\{ \begin{array}{ll} 0 & \text{if } l_j^x > -\infty, \\ -\infty & \text{otherwise,} \end{array} \right\} \quad \text{and} \quad \hat{u}_j^x := \left\{ \begin{array}{ll} 0 & \text{if } u_j^x < \infty, \\ \infty & \text{otherwise,} \end{array} \right\}$$

such that

$$c^T x < 0.$$

Such a solution implies that (12.5) is unbounded, and that (12.2) is infeasible.

In case that both the primal problem (12.1) and the dual problem (12.2) are infeasible, **MOSEK** will report only one of the two possible certificates — which one is not defined (**MOSEK** returns the first certificate found).

12.1.3 Minimalization vs. Maximalization

When the objective sense of problem (12.1) is maximization, i.e.

$$\begin{array}{lccccc} \text{maximize} & & & c^T x + c^f & & \\ \text{subject to} & l^c & \leq & Ax & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x, \end{array}$$

the objective sense of the dual problem changes to minimization, and the domain of all dual variables changes sign in comparison to (12.2). The dual problem thus takes the form

$$\begin{array}{ll}\text{minimize} & (l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x + c^f \\ \text{subject to} & \\ & A^T y + s_l^x - s_u^x = c, \\ & -y + s_l^c - s_u^c = 0, \\ & s_l^c, s_u^c, s_l^x, s_u^x \leq 0.\end{array}$$

This means that the duality gap, defined in (12.3) as the primal minus the dual objective value, becomes nonpositive. It follows that the dual objective will always be greater than or equal to the primal objective. The primal infeasibility certificate will be reported by **MOSEK** as a solution to the system

$$\begin{array}{l} A^T y + s_l^x - s_u^x = 0, \\ -y + s_l^c - s_u^c = 0, \\ s_l^c, s_u^c, s_l^x, s_u^x \leq 0, \end{array} \tag{12.6}$$

such that the objective value is strictly negative

$$(l^c)^T (s_l^c)^* - (u^c)^T (s_u^c)^* + (l^x)^T (s_l^x)^* - (u^x)^T (s_u^x)^* < 0.$$

Similarly, the certificate of dual infeasibility is an x satisfying the requirements of (12.5) such that $c^T x > 0$.

12.2 Conic Optimization

Conic optimization is an extension of linear optimization (see Sec. 12.1) allowing conic domains to be specified for affine expressions. A conic optimization problem to be solved by **MOSEK** can be written as

$$\begin{array}{lccccc}\text{minimize} & & & c^T x + c^f & & \\ \text{subject to} & l^c & \leq & Ax & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x, \\ & & & Fx + g & \in & \mathcal{D},\end{array} \tag{12.7}$$

where

- m is the number of constraints.
- n is the number of decision variables.
- $x \in \mathbb{R}^n$ is a vector of decision variables.
- $c \in \mathbb{R}^n$ is the linear part of the objective function.
- $c^f \in \mathbb{R}$ is a constant term in the objective
- $A \in \mathbb{R}^{m \times n}$ is the constraint matrix.
- $l^c \in \mathbb{R}^m$ is the lower limit on the activity for the constraints.
- $u^c \in \mathbb{R}^m$ is the upper limit on the activity for the constraints.
- $l^x \in \mathbb{R}^n$ is the lower limit on the activity for the variables.
- $u^x \in \mathbb{R}^n$ is the upper limit on the activity for the variables.

is the same as in Sec. 12.1 and moreover:

- $F \in \mathbb{R}^{k \times n}$ is the affine conic constraint matrix.,
- $g \in \mathbb{R}^k$ is the affine conic constraint constant term vector.,
- $\mathcal{D}$ is a Cartesian product of conic domains, namely $\mathcal{D} = \mathcal{D}_1 \times \cdots \times \mathcal{D}_p$, where p is the number of individual affine conic constraints (ACCs), and each domain is one from Sec. 15.11.

The total dimension of the domain $\mathcal{D}$ must be equal to k, the number of rows in F and g. Lower and upper bounds can be infinite, or in other words the corresponding bound may be omitted.

MOSEK supports also the cone of positive semidefinite matrices. In order not to obscure this section with additional notation, that extension is discussed in Sec. 12.3.

12.2.1 Duality for Conic Optimization

Corresponding to the primal problem (12.7), there is a dual problem

$$\begin{array}{ll} \text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x - g^T \dot{y} + c^f \\ \text{subject to} & \\ & \begin{array}{c} A^T y + s_l^x - s_u^x + F^T \dot{y} = c, \\ -y + s_l^c - s_u^c = 0, \\ s_l^c, s_u^c, s_l^x, s_u^x \geq 0, \\ \dot{y} \in \mathcal{D}^*, \end{array} \end{array} \tag{12.8}$$

where

- s_l^c are the dual variables for lower bounds of constraints,
- s_u^c are the dual variables for upper bounds of constraints,
- s_l^x are the dual variables for lower bounds of variables,
- s_u^x are the dual variables for upper bounds of variables,
- $\dot{y}$ are the dual variables for affine conic constraints,
- the dual domain $\mathcal{D}^* = \mathcal{D}_1^* \times \cdots \times \mathcal{D}_p^*$ is a Cartesian product of cones dual to $\mathcal{D}_i$.

One can check that the dual problem of the dual problem is identical to the original primal problem.

If a bound in the primal problem is plus or minus infinity, the corresponding dual variable is fixed at 0, and we use the convention that the product of the bound value and the corresponding dual variable is 0. This is equivalent to removing the corresponding dual variable $(s_l^x)_j$ from the dual problem. For example:

$$l_j^x = -\infty \quad \Rightarrow \quad (s_l^x)_j = 0 \text{ and } l_j^x \cdot (s_l^x)_j = 0.$$

A solution

$$(y, s_l^c, s_u^c, s_l^x, s_u^x, \dot{y})$$

to the dual problem is feasible if it satisfies all the constraints in (12.8). If (12.8) has at least one feasible solution, then (12.8) is *(dual) feasible*, otherwise the problem is *(dual) infeasible*.

A solution

$$(x^*, y^*, (s_l^c)^*, (s_u^c)^*, (s_l^x)^*, (s_u^x)^*, (\dot{y})^*)$$

is denoted a *primal-dual feasible solution*, if (x^*) is a solution to the primal problem (12.7) and $(y^*, (s_l^c)^*, (s_u^c)^*, (s_l^x)^*, (s_u^x)^*, (\dot{y})^*)$ is a solution to the corresponding dual problem (12.8). We also define an auxiliary vector

$$(x^c)^* := Ax^*$$

containing the activities of linear constraints.

For a primal-dual feasible solution we define the *duality gap* as the difference between the primal and the dual objective value,

$$\begin{array}{l} c^T x^* + c^f - \left\{ (l^c)^T (s_l^c)^* - (u^c)^T (s_u^c)^* + (l^x)^T (s_l^x)^* - (u^x)^T (s_u^x)^* - g^T (\dot{y})^* + c^f \right\} \\ = \sum_{i=0}^{m-1} \left[(s_l^c)_i^* ((x_i^c)^* - l_i^c) + (s_u^c)_i^* (u_i^c - (x_i^c)^*) \right] \\ + \sum_{j=0}^{n-1} \left[(s_l^x)_j^* (x_j - l_j^x) + (s_u^x)_j^* (u_j^x - x_j^*) \right] \\ + ((\dot{y})^*)^T (Fx^* + g) \geq 0 \end{array} \tag{12.9}$$

where the first relation can be obtained by transposing and multiplying the dual constraints (12.2) by x^* and $(x^c)^*$ respectively, and the second relation comes from the fact that each term in each sum is nonnegative. It follows that the primal objective will always be greater than or equal to the dual objective.

It is well-known that, under some non-degeneracy assumptions that exclude ill-posed cases, a conic optimization problem has an optimal solution if and only if there exist feasible primal-dual solution so that the duality gap is zero, or, equivalently, that the *complementarity conditions*

$$\begin{array}{rcll}
(s_l^c)_i^*((x_i^c)^* - l_i^c) & = & 0, & i = 0, \ldots, m-1, \\
(s_u^c)_i^*(u_i^c - (x_i^c)^*) & = & 0, & i = 0, \ldots, m-1, \\
(s_l^x)_j^*(x_j^* - l_j^x) & = & 0, & j = 0, \ldots, n-1, \\
(s_u^x)_j^*(u_j^x - x_j^*) & = & 0, & j = 0, \ldots, n-1, \\
((\dot{y})^*)^T(Fx^* + g) & = & 0, &
\end{array} \tag{12.10}$$

are satisfied.

If (12.7) has an optimal solution and **MOSEK** solves the problem successfully, both the primal and dual solution are reported, including a status indicating the exact state of the solution.

12.2.2 Infeasibility for Conic Optimization

Primal Infeasible Problems

If the problem (12.7) is infeasible (has no feasible solution), **MOSEK** will report a certificate of primal infeasibility: The dual solution reported is the certificate of infeasibility, and the primal solution is undefined.

A certificate of primal infeasibility is a feasible solution to the modified dual problem

$$\begin{array}{ll}
\text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x - g^T \dot{y} \\
\text{subject to} & \\
& \begin{array}{c}
A^T y + s_l^x - s_u^x + F^T \dot{y} = 0, \\
-y + s_l^c - s_u^c = 0, \\
s_l^c, s_u^c, s_l^x, s_u^x \geq 0, \\
\dot{y} \in \mathcal{D}^*,
\end{array}
\end{array} \tag{12.11}$$

such that the objective value is strictly positive, i.e. a solution

$$(y^*, (s_l^c)^*, (s_u^c)^*, (s_l^x)^*, (s_u^x)^*, (\dot{y})^*)$$

to (12.11) so that

$$(l^c)^T (s_l^c)^* - (u^c)^T (s_u^c)^* + (l^x)^T (s_l^x)^* - (u^x)^T (s_u^x)^* - g^T \dot{y} > 0.$$

Such a solution implies that (12.11) is unbounded, and that (12.7) is infeasible.

Dual Infeasible Problems

If the problem (12.8) is infeasible (has no feasible solution), **MOSEK** will report a certificate of dual infeasibility: The primal solution reported is the certificate of infeasibility, and the dual solution is undefined.

A certificate of dual infeasibility is a feasible solution to the modified primal problem

$$\begin{array}{lccccc}
\text{minimize} & & & c^T x & & \\
\text{subject to} & \hat{l}^c & \leq & Ax & \leq & \hat{u}^c, \\
& \hat{l}^x & \leq & x & \leq & \hat{u}^x, \\
& & & Fx & \in \mathcal{D} &
\end{array} \tag{12.12}$$

where

$$\hat{l}_i^c = \left\{ \begin{array}{ll} 0 & \text{if } l_i^c > -\infty, \\ -\infty & \text{otherwise,} \end{array} \right\} \quad \text{and} \quad \hat{u}_i^c := \left\{ \begin{array}{ll} 0 & \text{if } u_i^c < \infty, \\ \infty & \text{otherwise,} \end{array} \right\} \tag{12.13}$$

and

$$\hat{l}_j^x = \left\{ \begin{array}{ll} 0 & \text{if } l_j^x > -\infty, \\ -\infty & \text{otherwise,} \end{array} \right\} \quad \text{and} \quad \hat{u}_j^x := \left\{ \begin{array}{ll} 0 & \text{if } u_j^x < \infty, \\ \infty & \text{otherwise,} \end{array} \right\} \tag{12.14}$$

such that

$$c^T x < 0.$$

Such a solution implies that (12.12) is unbounded, and that (12.8) is infeasible.

In case that both the primal problem (12.7) and the dual problem (12.8) are infeasible, **MOSEK** will report only one of the two possible certificates — which one is not defined (**MOSEK** returns the first certificate found).

12.2.3 Minimalization vs. Maximalization

When the objective sense of problem (12.7) is maximization, i.e.

$$\begin{array}{lccccc} \text{maximize} & & & c^T x + c^f & & \\ \text{subject to} & l^c & \leq & Ax & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x, \\ & & & Fx + g & \in & \mathcal{D}, \end{array}$$

the objective sense of the dual problem changes to minimization, and the domain of all dual variables changes sign in comparison to (12.2). The dual problem thus takes the form

$$\begin{array}{ll} \text{minimize} & (l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x - g^T \dot{y} + c^f \\ \text{subject to} & A^T y + s_l^x - s_u^x + F^T \dot{y} = c, \\ & -y + s_l^c - s_u^c = 0, \\ & s_l^c, s_u^c, s_l^x, s_u^x \leq 0, \\ & -\dot{y} \in \mathcal{D}^* \end{array}$$

This means that the duality gap, defined in (12.9) as the primal minus the dual objective value, becomes nonpositive. It follows that the dual objective will always be greater than or equal to the primal objective. The primal infeasibility certificate will be reported by **MOSEK** as a solution to the system

$$\begin{array}{c} A^T y + s_l^x - s_u^x + F^T \dot{y} = 0, \\ -y + s_l^c - s_u^c = 0, \\ s_l^c, s_u^c, s_l^x, s_u^x \leq 0, \\ -\dot{y} \in \mathcal{D}^* \end{array} \tag{12.15}$$

such that the objective value is strictly negative

$$(l^c)^T (s_l^c)^* - (u^c)^T (s_u^c)^* + (l^x)^T (s_l^x)^* - (u^x)^T (s_u^x)^* - g^T \dot{y} < 0.$$

Similarly, the certificate of dual infeasibility is an x satisfying the requirements of (12.12) such that $c^T x > 0$.

12.3 Semidefinite Optimization

Semidefinite optimization is an extension of conic optimization (see Sec. 12.2) allowing positive semidefinite matrix variables to be used in addition to the usual scalar variables. All the other parts of the input are defined exactly as in Sec. 12.2, and the discussion from that section applies verbatim to all properties of problems with semidefinite variables. We only briefly indicate how the corresponding formulae should be modified with semidefinite terms.

A semidefinite optimization problem can be written as

$$\begin{array}{lccccc} \text{minimize} & & & c^T x + \langle \overline{C}, \overline{X} \rangle + c^f & & \\ \text{subject to} & l^c & \leq & Ax + \langle \overline{A}, \overline{X} \rangle & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x, \\ & & & Fx + \langle \overline{F}, \overline{X} \rangle + g & \in & \mathcal{D}, \\ & & & \overline{X}_j & \in & \mathcal{S}_+^{r_j}, j = 1, \ldots, s \end{array}$$

where

- m is the number of constraints.
- n is the number of decision variables.
- $x \in \mathbb{R}^n$ is a vector of decision variables.
- $c \in \mathbb{R}^n$ is the linear part of the objective function.
- $c^f \in \mathbb{R}$ is a constant term in the objective
- $A \in \mathbb{R}^{m \times n}$ is the constraint matrix.
- $l^c \in \mathbb{R}^m$ is the lower limit on the activity for the constraints.
- $u^c \in \mathbb{R}^m$ is the upper limit on the activity for the constraints.
- $l^x \in \mathbb{R}^n$ is the lower limit on the activity for the variables.
- $u^x \in \mathbb{R}^n$ is the upper limit on the activity for the variables.
- $F \in \mathbb{R}^{k \times n}$ is the affine conic constraint matrix.,
- $g \in \mathbb{R}^k$ is the affine conic constraint constant term vector.,
- $\mathcal{D}$ is a Cartesian product of conic domains, namely $\mathcal{D} = \mathcal{D}_1 \times \cdots \times \mathcal{D}_p$, where p is the number of individual affine conic constraints (ACCs), and each domain is one from Sec. 15.11.

is the same as in Sec. 12.2 and moreover:

- there are s symmetric positive semidefinite variables, the j-th of which is $\overline{X}_j \in \mathcal{S}_+^{r_j}$ of dimension r_j,
- $\overline{C} = (\overline{C}_j)_{j=1,\ldots,s}$ is a collection of symmetric coefficient matrices in the objective, with $\overline{C}_j \in \mathcal{S}^{r_j}$, and we interpret the notation $\langle \overline{C}, \overline{X} \rangle$ as a shorthand for

$$\langle \overline{C}, \overline{X} \rangle := \sum_{j=1}^{s} \langle \overline{C}_j, \overline{X}_j \rangle.$$

- $\overline{A} = (\overline{A}_{ij})_{i=1,\ldots,m, j=1,\ldots,s}$ is a collection of symmetric coefficient matrices in the constraints, with $\overline{A}_{ij} \in \mathcal{S}^{r_j}$, and we interpret the notation $\langle \overline{A}, \overline{X} \rangle$ as a shorthand for the vector

$$\langle \overline{A}, \overline{X} \rangle := \left(\sum_{j=1}^{s} \langle \overline{A}_{ij}, \overline{X}_j \rangle \right)_{i=1,\ldots,m}.$$

- $\overline{F} = (\overline{F}_{ij})_{i=1,\ldots,k, j=1,\ldots,s}$ is a collection of symmetric coefficient matrices in the affine conic constraints, with $\overline{F}_{ij} \in \mathcal{S}^{r_j}$, and we interpret the notation $\langle \overline{F}, \overline{X} \rangle$ as a shorthand for the vector

$$\langle \overline{F}, \overline{X} \rangle := \left(\sum_{j=1}^{s} \langle \overline{F}_{ij}, \overline{X}_j \rangle \right)_{i=1,\ldots,k}.$$

In each case the matrix inner product between symmetric matrices of the same dimension r is defined as

$$\langle U, V \rangle := \sum_{i=1}^{r} \sum_{j=1}^{r} U_{ij} V_{ij}.$$

To summarize, above the formulation extends that from Sec. 12.2 by the possibility of including semidefinite terms in the objective, constraints and affine conic constraints.

Duality

The definition of the dual problem (12.8) becomes:

$$\begin{array}{ll}\text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x - g^T \dot{y} + c^f \\ \text{subject to} & \\ & \begin{array}{ll} A^T y + s_l^x - s_u^x + F^T \dot{y} = c, & \\ -y + s_l^c - s_u^c = 0, & \\ \overline{C}_j - \sum_{i=1}^m y_i \overline{A}_{ij} - \sum_{i=1}^k \dot{y}_i \overline{F}_{ij} = S_j, & j = 1, \ldots, s, \\ s_l^c, s_u^c, s_l^x, s_u^x \geq 0, & \\ \dot{y} \in \mathcal{D}^*, & \\ \overline{S}_j \in \mathcal{S}_+^{r_j}, & j = 1, \ldots, s. \end{array} \end{array} \tag{12.16}$$

Complementarity conditions (12.10) include the additional relation:

$$\langle \overline{X}_j, \overline{S}_j \rangle = 0 \quad j = 1, \ldots, s. \tag{12.17}$$

Infeasibility

A certificate of primal infeasibility (12.11) is now a feasible solution to:

$$\begin{array}{ll}\text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x - g^T \dot{y} \\ \text{subject to} & \\ & \begin{array}{ll} A^T y + s_l^x - s_u^x + F^T \dot{y} = 0, & \\ -y + s_l^c - s_u^c = 0, & \\ -\sum_{i=1}^m y_i \overline{A}_{ij} - \sum_{i=1}^k \dot{y}_i \overline{F}_{ij} = S_j, & j = 1, \ldots, s, \\ s_l^c, s_u^c, s_l^x, s_u^x \geq 0, & \\ \dot{y} \in \mathcal{D}^*, & \\ \overline{S}_j \in \mathcal{S}_+^{r_j}, & j = 1, \ldots, s. \end{array} \end{array} \tag{12.18}$$

such that the objective value is strictly positive.

Similarly, a dual infeasibility certificate (12.12) is a feasible solution to

$$\begin{array}{lccccl} \text{minimize} & & & c^T x + \langle \overline{C}, \overline{X} \rangle & & \\ \text{subject to} & \hat{l}^c & \leq & Ax + \langle \overline{A}, \overline{X} \rangle & \leq & \hat{u}^c, \\ & \hat{l}^x & \leq & x & \leq & \hat{u}^x, \\ & & & Fx + \langle \overline{F}, \overline{X} \rangle & \in & \mathcal{D}, \\ & & & \overline{X}_j & \in & \mathcal{S}_+^{r_j}, j = 1, \ldots, s \end{array} \tag{12.19}$$

where the modified bounds are as in (12.13) and (12.14) and the objective value is strictly negative.

12.4 Quadratic and Quadratically Constrained Optimization

A convex quadratic and quadratically constrained optimization problem has the form

$$\begin{array}{lccccll} \text{minimize} & & & \frac{1}{2} x^T Q^o x + c^T x + c^f & & & \\ \text{subject to} & l_k^c & \leq & \frac{1}{2} x^T Q^k x + \sum_{j=0}^{n-1} a_{kj} x_j & \leq & u_k^c, & k = 0, \ldots, m-1, \\ & l_j^x & \leq & x_j & \leq & u_j^x, & j = 0, \ldots, n-1, \end{array} \tag{12.20}$$

where all variables and bounds have the same meaning as for linear problems (see Sec. 12.1) and Q^o and all Q^k are symmetric matrices. Moreover, for convexity, Q^o must be a positive semidefinite matrix and Q^k must satisfy

$$\begin{array}{rcl} -\infty < l_k^c & \Rightarrow & Q^k \text{ is negative semidefinite,} \\ u_k^c < \infty & \Rightarrow & Q^k \text{ is positive semidefinite,} \\ -\infty < l_k^c \leq u_k^c < \infty & \Rightarrow & Q^k = 0. \end{array}$$

The convexity requirement is very important and **MOSEK** checks whether it is fulfilled.

12.4.1 A Recommendation

Any convex quadratic optimization problem can be reformulated as a conic quadratic optimization problem, see Modeling Cookbook and [And13]. In fact **MOSEK** does such conversion internally as a part of the solution process for the following reasons:

- the conic optimizer is numerically more robust than the one for quadratic problems.
- the conic optimizer is usually faster because quadratic cones are simpler than quadratic functions, even though the conic reformulation usually has more constraints and variables than the original quadratic formulation.
- it is easy to dualize the conic formulation if deemed worthwhile potentially leading to (huge) computational savings.

However, instead of relying on the automatic reformulation we recommend to formulate the problem as a conic problem from scratch because:

- it saves the computational overhead of the reformulation including the convexity check. A conic problem is convex by construction and hence no convexity check is needed for conic problems.
- usually the modeler can do a better reformulation than the automatic method because the modeler can exploit the knowledge of the problem at hand.

To summarize we recommend to formulate quadratic problems and in particular quadratically constrained problems directly in conic form.

12.4.2 Duality for Quadratic and Quadratically Constrained Optimization

The dual problem corresponding to the quadratic and quadratically constrained optimization problem (12.20) is given by

$$\begin{array}{lc} \text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x + \frac{1}{2} x^T \left\{ \sum_{k=0}^{m-1} y_k Q^k - Q^o \right\} x + c^f \\ \text{subject to} & A^T y + s_l^x - s_u^x + \left\{ \sum_{k=0}^{m-1} y_k Q^k - Q^o \right\} x = c, \\ & -y + s_l^c - s_u^c = 0, \\ & s_l^c, s_u^c, s_l^x, s_u^x \geq 0. \end{array} \tag{12.21}$$

The dual problem is related to the dual problem for linear optimization (see Sec. 12.1.1), but depends on the variable x which in general can not be eliminated. In the solutions reported by **MOSEK**, the value of x is the same for the primal problem (12.20) and the dual problem (12.21).

12.4.3 Infeasibility for Quadratic Optimization

In case **MOSEK** finds a problem to be infeasible it reports a certificate of infeasibility. We write them out explicitly for quadratic problems, that is when $Q^k = 0$ for all k and quadratic terms appear only in the objective Q^o. In this case the constraints both in the primal and dual problem are linear, and **MOSEK** produces for them the same infeasibility certificate as for linear problems.

The certificate of primal infeasibility is a solution to the problem (12.4) such that the objective value is strictly positive.

The certificate of dual infeasibility is a solution to the problem (12.5) together with an additional constraint

$$Q^o x = 0$$

such that the objective value is strictly negative.

Below is an outline of the different problem types for quick reference.

Continuous problem formulations

- **Linear optimization (LO)**

$$\begin{array}{lccccc} \text{minimize} & & & c^T x + c^f & & \\ \text{subject to} & l^c & \leq & Ax & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x. \end{array}$$

- **Conic optimization (CO)**

  Conic optimization extends linear optimization with *affine conic constraints* (ACC):

$$\begin{array}{lccccc} \text{minimize} & & & c^T x + c^f & & \\ \text{subject to} & l^c & \leq & Ax & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x, \\ & & & Fx + g & \in & \mathcal{D}, \end{array}$$

  where $\mathcal{D}$ is a product of domains from Sec. 15.11.

- **Semidefinite optimization (SDO)**

  A conic optimization problem can be further extended with *semidefinite variables*:

$$\begin{array}{lccccc} \text{minimize} & & & c^T x + \langle \overline{C}, \overline{X} \rangle + c^f & & \\ \text{subject to} & l^c & \leq & Ax + \langle \overline{A}, \overline{X} \rangle & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x, \\ & & & Fx + \langle \overline{F}, \overline{X} \rangle + g & \in & \mathcal{D}, \\ & & & \overline{X} & \in & \mathcal{S}_+, \end{array}$$

  where $\mathcal{D}$ is a product of domains from Sec. 15.11 and $\mathcal{S}_+$ is a product of PSD cones meaning that $\overline{X}$ is a sequence of PSD matrix variables.

- **Quadratic and quadratically constrained optimization (QO, QCQO)**

  A quadratic problem or quadratically constrained problem has the form

$$\begin{array}{lccccc} \text{minimize} & & & \frac{1}{2} x^T Q^o x + c^T x + c^f & & \\ \text{subject to} & l^c & \leq & \frac{1}{2} x^T Q^c x + Ax & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x. \end{array}$$

Mixed-integer extensions

Coninuous problems can be extended with constraints requiring the mixed-integer optimizer. We outline them briefly here. The continuous part of a mixed-integer problem is formulated according to one of the continuous types above, however only the primal information and solution fields are relevant, there are no dual values and no infeasibility certificates.

- **Integer variables.** Specifies that a subset of variables take integer values, that is

$$x_I \in \mathbb{Z}$$

  for some index set I.

- **Disjunctive constraints.** Appends disjunctions of the form

$$\bigvee_{i=1}^{t} \bigwedge_{j=1}^{s_i} (D_{ij} x + d_{ij} \in \mathcal{D}_{ij})$$

  ie. a disjunction of conjunctions of linear constraints, where each $D_{ij} x + d_{ij}$ is an affine expression of the optimization variables and each $\mathcal{D}_{ij}$ is an affine domain. Linear and conic problems can be extended with disjunctive constraints.

Chapter 13

Optimizers

The most essential part of **MOSEK** are the optimizers:

- *primal simplex* (linear problems),
- *dual simplex* (linear problems),
- *interior-point* (linear, quadratic and conic problems),
- *mixed-integer* (problems with integer variables).

The structure of a successful optimization process is roughly:

- **Presolve**
  1. *Elimination*: Reduce the size of the problem.
  2. *Dualizer*: Choose whether to solve the primal or the dual form of the problem.
  3. *Scaling*: Scale the problem for better numerical stability.
- **Optimization**
  1. *Optimize*: Solve the problem using selected method.
  2. *Terminate*: Stop the optimization when specific termination criteria have been met.
  3. *Report*: Return the solution or an infeasibility certificate.

The preprocessing stage is transparent to the user, but useful to know about for tuning purposes. The purpose of the preprocessing steps is to make the actual optimization more efficient and robust. We discuss the details of the above steps in the following sections.

13.1 Presolve

Before an optimizer actually performs the optimization the problem is preprocessed using the so-called presolve. The purpose of the presolve is to

1. remove redundant constraints,
2. eliminate fixed variables,
3. remove linear dependencies,
4. substitute out (implied) free variables, and
5. reduce the size of the optimization problem in general.

After the presolved problem has been optimized the solution is automatically postsolved so that the returned solution is valid for the original problem. Hence, the presolve is completely transparent. For further details about the presolve phase, please see [AA95] and [AGMeszarosX96].

It is possible to fine-tune the behavior of the presolve or to turn it off entirely. If presolve consumes too much time or memory compared to the reduction in problem size gained it may be disabled. This is done by setting the parameter *iparam.presolve_use* to *presolvemode.off*.

In the following we describe in more detail the presolve applied to continuous, i.e., linear and conic optimization problems, see Sec. 13.2 and Sec. 13.3. The mixed-integer optimizer, Sec. 13.4, applies similar techniques. The two most time-consuming steps of the presolve for continuous optimization problems are

- the eliminator, and
- the linear dependency check.

Therefore, in some cases it is worthwhile to disable one or both of these.

Numerical issues in the presolve

During the presolve the problem is reformulated so that it hopefully solves faster. However, in rare cases the presolved problem may be harder to solve then the original problem. The presolve may also be infeasible although the original problem is not. If it is suspected that presolved problem is much harder to solve than the original, we suggest to first turn the eliminator off by setting the parameter *iparam.presolve_eliminator_max_num_tries* to 0. If that does not help, then trying to turn entire presolve off may help.

Since all computations are done in finite precision, the presolve employs some tolerances when concluding a variable is fixed or a constraint is redundant. If it happens that **MOSEK** incorrectly concludes a problem is primal or dual infeasible, then it is worthwhile to try to reduce the parameters *dparam.presolve_tol_x* and *dparam.presolve_tol_s*. However, if reducing the parameters actually helps then this should be taken as an indication that the problem is badly formulated.

Eliminator

The purpose of the eliminator is to eliminate free and implied free variables from the problem using substitution. For instance, given the constraints

$$\begin{array}{rcl} y & = & \sum_j x_j, \\ y, x & \geq & 0, \end{array}$$

y is an implied free variable that can be substituted out of the problem, if deemed worthwhile. If the eliminator consumes too much time or memory compared to the reduction in problem size gained it may be disabled. This can be done by setting the parameter *iparam.presolve_eliminator_max_num_tries* to 0. In rare cases the eliminator may cause that the problem becomes much hard to solve.

Linear dependency checker

The purpose of the linear dependency check is to remove linear dependencies among the linear equalities. For instance, the three linear equalities

$$\begin{array}{rcl} x_1 + x_2 + x_3 & = & 1, \\ x_1 + 0.5x_2 & = & 0.5, \\ 0.5x_2 + x_3 & = & 0.5. \end{array}$$

contain exactly one linear dependency. This implies that one of the constraints can be dropped without changing the set of feasible solutions. Removing linear dependencies is in general a good idea since it reduces the size of the problem. Moreover, the linear dependencies are likely to introduce numerical problems in the optimization phase. It is best practice to build models without linear dependencies, but that is not always easy for the user to control. If the linear dependencies are removed at the modeling stage, the linear dependency check can safely be disabled by setting the parameter *iparam.presolve_lindep_use* to *onoffkey.off*.

Dualizer

All linear, conic, and convex optimization problems have an equivalent dual problem associated with them. **MOSEK** has built-in heuristics to determine if it is more efficient to solve the primal or dual problem. The form (primal or dual) is displayed in the **MOSEK** log and available as an information item from the solver. Should the internal heuristics not choose the most efficient form of the problem it may be worthwhile to set the dualizer manually by setting the parameters:

- *iparam.intpnt_solve_form*: In case of the interior-point optimizer.
- *iparam.sim_solve_form*: In case of the simplex optimizer.

Note that currently only linear and conic (but not semidefinite) problems may be automatically dualized.

Scaling

Problems containing data with large and/or small coefficients, say $1.0e+9$ or $1.0e-7$, are often hard to solve. Significant digits may be truncated in calculations with finite precision, which can result in the optimizer relying on inaccurate data. Since computers work in finite precision, extreme coefficients should be avoided. In general, data around the same *order of magnitude* is preferred, and we will refer to a problem, satisfying this loose property, as being *well-scaled*. If the problem is not well scaled, **MOSEK** will try to scale (multiply) constraints and variables by suitable constants. **MOSEK** solves the scaled problem to improve the numerical properties.

The scaling process is transparent, i.e. the solution to the original problem is reported. It is important to be aware that the optimizer terminates when the termination criterion is met on the scaled problem, therefore significant primal or dual infeasibilities may occur after unscaling for badly scaled problems. The best solution of this issue is to reformulate the problem, making it better scaled.

By default **MOSEK** heuristically chooses a suitable scaling. The scaling for interior-point and simplex optimizers can be controlled with the parameters *iparam.intpnt_scaling* and *iparam.sim_scaling* respectively.

13.2 Linear Optimization

13.2.1 Optimizer Selection

Two different types of optimizers are available for linear problems: The default is an interior-point method, and the alternative is the simplex method (primal or dual). The optimizer can be selected using the parameter *iparam.optimizer*.

The Interior-point or the Simplex Optimizer?

Given a linear optimization problem, which optimizer is the best: the simplex or the interior-point optimizer? It is impossible to provide a general answer to this question. However, the interior-point optimizer behaves more predictably: it tends to use between 20 and 100 iterations, almost independently of problem size, but cannot perform warm-start. On the other hand the simplex method can take advantage of an initial solution, but is less predictable from cold-start. The interior-point optimizer is used by default.

The Primal or the Dual Simplex Variant?

MOSEK provides both a primal and a dual simplex optimizer. Predicting which simplex optimizer is faster is impossible, however, in recent years the dual optimizer has seen several algorithmic and computational improvements, which, in our experience, make it faster on average than the primal version. Still, it depends much on the problem structure and size. Setting the *iparam.optimizer* parameter to *optimizertype.free_simplex* instructs **MOSEK** to choose one of the simplex variants automatically.

To summarize, if you want to know which optimizer is faster for a given problem type, it is best to try all the options.

13.2.2 The Interior-point Optimizer

The purpose of this section is to provide information about the algorithm employed in the **MOSEK** interior-point optimizer for linear problems and about its termination criteria.

The homogeneous primal-dual problem

In order to keep the discussion simple it is assumed that **MOSEK** solves linear optimization problems of standard form

$$\begin{array}{lll} \text{minimize} & c^T x & \\ \text{subject to} & Ax & = b, \\ & x \geq 0. & \end{array} \tag{13.1}$$

This is in fact what happens inside **MOSEK**; for efficiency reasons **MOSEK** converts the problem to standard form before solving, then converts it back to the input form when reporting the solution.

Since it is not known beforehand whether problem (13.1) has an optimal solution, is primal infeasible or is dual infeasible, the optimization algorithm must deal with all three situations. This is the reason why **MOSEK** solves the so-called homogeneous model

$$\begin{array}{rcl} Ax - b\tau & = & 0, \\ A^T y + s - c\tau & = & 0, \\ -c^T x + b^T y - \kappa & = & 0, \\ x, s, \tau, \kappa & \geq & 0, \end{array} \tag{13.2}$$

where y and s correspond to the dual variables in (13.1), and τ and κ are two additional scalar variables. Note that the homogeneous model (13.2) always has solution since

$$(x, y, s, \tau, \kappa) = (0, 0, 0, 0, 0)$$

is a solution, although not a very interesting one. Any solution

$$(x^*, y^*, s^*, \tau^*, \kappa^*)$$

to the homogeneous model (13.2) satisfies

$$x_j^* s_j^* = 0 \text{ and } \tau^* \kappa^* = 0.$$

Moreover, there is always a solution that has the property $\tau^* + \kappa^* > 0$.

First, assume that $\tau^* > 0$. It follows that

$$\begin{array}{rcl} A\frac{x^*}{\tau^*} & = & b, \\ A^T \frac{y^*}{\tau^*} + \frac{s^*}{\tau^*} & = & c, \\ -c^T \frac{x^*}{\tau^*} + b^T \frac{y^*}{\tau^*} & = & 0, \\ x^*, s^*, \tau^*, \kappa^* & \geq & 0. \end{array}$$

This shows that $\frac{x^*}{\tau^*}$ is a primal optimal solution and $(\frac{y^*}{\tau^*}, \frac{s^*}{\tau_*})$ is a dual optimal solution; this is reported as the optimal interior-point solution since

$$(x, y, s) = \left\{ \frac{x^*}{\tau^*}, \frac{y^*}{\tau^*}, \frac{s^*}{\tau^*} \right\}$$

is a primal-dual optimal solution (see Sec. 12.1 for the mathematical background on duality and optimality).

On other hand, if $\kappa^* > 0$ then

$$\begin{array}{rcl} Ax^* & = & 0, \\ A^T y^* + s^* & = & 0, \\ -c^T x^* + b^T y^* & = & \kappa^*, \\ x^*, s^*, \tau^*, \kappa^* & \geq & 0. \end{array}$$

This implies that at least one of

$$c^T x^* < 0 \tag{13.3}$$

or

$$b^T y^* > 0 \tag{13.4}$$

is satisfied. If (13.3) is satisfied then x^* is a certificate of dual infeasibility, whereas if (13.4) is satisfied then y^* is a certificate of primal infeasibility.

In summary, by computing an appropriate solution to the homogeneous model, all information required for a solution to the original problem is obtained. A solution to the homogeneous model can be computed using a primal-dual interior-point algorithm [And09].

Interior-point Termination Criterion

For efficiency reasons it is not practical to solve the homogeneous model exactly. Hence, an exact optimal solution or an exact infeasibility certificate cannot be computed and a reasonable termination criterion has to be employed.

In the k-th iteration of the interior-point algorithm a trial solution

$$(x^k, y^k, s^k, \tau^k, \kappa^k)$$

to homogeneous model is generated, where

$$x^k, s^k, \tau^k, \kappa^k > 0.$$

Optimal case

Whenever the trial solution satisfies the criterion

$$\begin{array}{rcl} \left\| A\frac{x^k}{\tau^k} - b \right\|_\infty & \leq & \epsilon_p(1 + \|b\|_\infty), \\ \left\| A^T\frac{y^k}{\tau^k} + \frac{s^k}{\tau^k} - c \right\|_\infty & \leq & \epsilon_d(1 + \|c\|_\infty), \text{ and} \\ \min\left(\frac{(x^k)^T s^k}{(\tau^k)^2}, |\frac{c^T x^k}{\tau^k} - \frac{b^T y^k}{\tau^k}|\right) & \leq & \epsilon_g \max\left(1, \frac{\min(|c^T x^k|, |b^T y^k|)}{\tau^k}\right), \end{array} \tag{13.5}$$

the interior-point optimizer is terminated and

$$\frac{(x^k, y^k, s^k)}{\tau^k}$$

is reported as the primal-dual optimal solution. The interpretation of (13.5) is that the optimizer is terminated if

- $\frac{x^k}{\tau^k}$ is approximately primal feasible,
- $\left\{\frac{y^k}{\tau^k}, \frac{s^k}{\tau^k}\right\}$ is approximately dual feasible, and
- the duality gap is almost zero.

Dual infeasibility certificate

On the other hand, if the trial solution satisfies

$$-\epsilon_i c^T x^k > \frac{\|c\|_\infty}{\max(1, \|b\|_\infty)} \left\| Ax^k \right\|_\infty$$

then the problem is declared dual infeasible and x^k is reported as a certificate of dual infeasibility. The motivation for this stopping criterion is as follows: First assume that $\left\| Ax^k \right\|_\infty = 0$; then x^k is an exact certificate of dual infeasibility. Next assume that this is not the case, i.e.

$$\left\| Ax^k \right\|_\infty > 0,$$

and define

$$\bar{x} := \epsilon_i \frac{\max(1, \|b\|_\infty)}{\|Ax^k\|_\infty \|c\|_\infty} x^k.$$

It is easy to verify that

$$\|A\bar{x}\|_\infty = \epsilon_i \frac{\max(1, \|b\|_\infty)}{\|c\|_\infty} \text{ and } -c^T\bar{x} > 1,$$

which shows $\bar{x}$ is an approximate certificate of dual infeasibility, where ε_i controls the quality of the approximation. A smaller value means a better approximation.

Primal infeasibility certificate

Finally, if

$$\epsilon_i b^T y^k > \frac{\|b\|_\infty}{\max(1, \|c\|_\infty)} \left\|A^T y^k + s^k\right\|_\infty$$

then y^k is reported as a certificate of primal infeasibility.

Adjusting optimality criteria

It is possible to adjust the tolerances ε_p, ε_d, ε_g and ε_i using parameters; see table for details.

Table 13.1: Parameters employed in termination criterion

ToleranceParameter	name
ε_p	*dparam.intpnt_tol_pfeas*
ε_d	*dparam.intpnt_tol_dfeas*
ε_g	*dparam.intpnt_tol_rel_gap*
ε_i	*dparam.intpnt_tol_infeas*

The default values of the termination tolerances are chosen such that for a majority of problems appearing in practice it is not possible to achieve much better accuracy. Therefore, tightening the tolerances usually is not worthwhile. However, an inspection of (13.5) reveals that the quality of the solution depends on $\|b\|_\infty$ and $\|c\|_\infty$; the smaller the norms are, the better the solution accuracy.

The interior-point method as implemented by **MOSEK** will converge toward optimality and primal and dual feasibility at the same rate [And09]. This means that if the optimizer is stopped prematurely then it is very unlikely that either the primal or dual solution is feasible. Another consequence is that in most cases all the tolerances, ε_p, ε_d, ε_g and ε_i, have to be relaxed together to achieve an effect.

The basis identification discussed in Sec. 13.2.2 requires an optimal solution to work well; hence basis identification should be turned off if the termination criterion is relaxed.

To conclude the discussion in this section, relaxing the termination criterion is usually not worthwhile.

Basis Identification

An interior-point optimizer does not return an optimal basic solution unless the problem has a unique primal and dual optimal solution. Therefore, the interior-point optimizer has an optional post-processing step that computes an optimal basic solution starting from the optimal interior-point solution. More information about the basis identification procedure may be found in [AY96]. In the following we provide an overall idea of the procedure.

There are some cases in which a basic solution could be more valuable:

- a basic solution is often more accurate than an interior-point solution,
- a basic solution can be used to warm-start the simplex algorithm in case of reoptimization,
- a basic solution is in general more sparse, i.e. more variables are fixed to zero. This is particularly appealing when solving continuous relaxations of mixed integer problems, as well as in all applications in which sparser solutions are preferred.

To illustrate how the basis identification routine works, we use the following trivial example:

$$\begin{array}{lll} \text{minimize} & x+y & \\ \text{subject to} & x+y & = \quad 1, \\ & x,y \geq 0. & \end{array}$$

It is easy to see that all feasible solutions are also optimal. In particular, there are two basic solutions, namely

$$\begin{array}{rcl} (x_1^*, y_1^*) & = & (1,0), \\ (x_2^*, y_2^*) & = & (0,1). \end{array}$$

The interior point algorithm will actually converge to the center of the optimal set, i.e. to $(x^*, y^*) = (1/2, 1/2)$ (to see this in **MOSEK** deactivate *Presolve*).

In practice, when the algorithm gets close to the optimal solution, it is possible to construct in polynomial time an initial basis for the simplex algorithm from the current interior point solution. This basis is used to warm-start the simplex algorithm that will provide the optimal basic solution. In most cases the constructed basis is optimal, or very few iterations are required by the simplex algorithm to make it optimal and hence the final *clean-up* phase be short. However, for some cases of ill-conditioned problems the additional simplex clean up phase may take of lot a time.

By default **MOSEK** performs a basis identification. However, if a basic solution is not needed, the basis identification procedure can be turned off. The parameters

- *iparam.intpnt_basis*,
- *iparam.bi_ignore_max_iter*, and
- *iparam.bi_ignore_num_error*

control when basis identification is performed.

The type of simplex algorithm to be used (primal/dual) can be tuned with the parameter *iparam.bi_clean_optimizer*, and the maximum number of iterations can be set with *iparam.bi_max_iterations*.

Finally, it should be mentioned that there is no guarantee on which basic solution will be returned.

The Interior-point Log

Below is a typical log output from the interior-point optimizer:

```
Optimizer  - threads                : 1
Optimizer  - solved problem         : the dual
Optimizer  - Constraints            : 2
Optimizer  - Cones                  : 0
Optimizer  - Scalar variables       : 6                 conic                  : 0
Optimizer  - Semi-definite variables: 0                 scalarized             : 0
Factor     - setup time             : 0.00              dense det. time        : 0.00
Factor     - ML order time          : 0.00              GP order time          : 0.00
Factor     - nonzeros before factor : 3                 after factor           : 3
Factor     - dense dim.             : 0                 flops                  : 7.
↪00e+001
ITE PFEAS    DFEAS    GFEAS    PRSTATUS   POBJ              DOBJ              MU
↪ TIME
0   1.0e+000 8.6e+000 6.1e+000 1.00e+000  0.000000000e+000  -2.208000000e+003 1.
↪0e+000 0.00
1   1.1e+000 2.5e+000 1.6e-001 0.00e+000  -7.901380925e+003 -7.394611417e+003 2.
↪5e+000 0.00
2   1.4e-001 3.4e-001 2.1e-002 8.36e-001  -8.113031650e+003 -8.055866001e+003 3.3e-
↪001 0.00
3   2.4e-002 5.8e-002 3.6e-003 1.27e+000  -7.777530698e+003 -7.766471080e+003 5.7e-
↪002 0.01
```

(continues on next page)

(continued from previous page)

```
4   1.3e-004 3.2e-004 2.0e-005 1.08e+000  -7.668323435e+003 -7.668207177e+003 3.2e-
↪004 0.01
5   1.3e-008 3.2e-008 2.0e-009 1.00e+000  -7.668000027e+003 -7.668000015e+003 3.2e-
↪008 0.01
6   1.3e-012 3.2e-012 2.0e-013 1.00e+000  -7.667999994e+003 -7.667999994e+003 3.2e-
↪012 0.01
```

The first line displays the number of threads used by the optimizer and the second line indicates if the optimizer chose to solve the primal or dual problem (see *iparam.intpnt_solve_form*). The next lines display the problem dimensions as seen by the optimizer, and the `Factor...` lines show various statistics. This is followed by the iteration log.

Using the same notation as in Sec. 13.2.2 the columns of the iteration log have the following meaning:

- `ITE`: Iteration index k.
- `PFEAS`: $\left\|Ax^k - b\tau^k\right\|_\infty$. The numbers in this column should converge monotonically towards zero but may stall at low level due to rounding errors.
- `DFEAS`: $\left\|A^T y^k + s^k - c\tau^k\right\|_\infty$. The numbers in this column should converge monotonically towards zero but may stall at low level due to rounding errors.
- `GFEAS`: $|-c^T x^k + b^T y^k - \kappa^k|$. The numbers in this column should converge monotonically towards zero but may stall at low level due to rounding errors.
- `PRSTATUS`: This number converges to 1 if the problem has an optimal solution whereas it converges to -1 if that is not the case.
- `POBJ`: $c^T x^k/\tau^k$. An estimate for the primal objective value.
- `DOBJ`: $b^T y^k/\tau^k$. An estimate for the dual objective value.
- `MU`: $\frac{(x^k)^T s^k + \tau^k \kappa^k}{n+1}$. The numbers in this column should always converge to zero.
- `TIME`: Time spent since the optimization started.

13.2.3 The Simplex Optimizer

An alternative to the interior-point optimizer is the simplex optimizer. The simplex optimizer uses a different method that allows exploiting an initial guess for the optimal solution to reduce the solution time. Depending on the problem it may be faster or slower to use an initial guess; see Sec. 13.2.1 for a discussion. **MOSEK** provides both a primal and a dual variant of the simplex optimizer.

Simplex Termination Criterion

The simplex optimizer terminates when it finds an optimal basic solution or an infeasibility certificate. A basic solution is optimal when it is primal and dual feasible; see Sec. 12.1 for a definition of the primal and dual problem. Due to the fact that computations are performed in finite precision **MOSEK** allows violations of primal and dual feasibility within certain tolerances. The user can control the allowed primal and dual tolerances with the parameters *dparam.basis_tol_x* and *dparam.basis_tol_s*.

Setting the parameter *iparam.optimizer* to *optimizertype.free_simplex* instructs **MOSEK** to select automatically between the primal and the dual simplex optimizers. Hence, **MOSEK** tries to choose the best optimizer for the given problem and the available solution. The same parameter can also be used to force one of the variants.

Starting From an Existing Solution

When using the simplex optimizer it may be possible to reuse an existing solution and thereby reduce the solution time significantly. When a simplex optimizer starts from an existing solution it is said to perform a *warm-start*. If the user is solving a sequence of optimization problems by solving the problem, making modifications, and solving again, **MOSEK** will warm-start automatically.

By default **MOSEK** uses presolve when performing a warm-start. If the optimizer only needs very few iterations to find the optimal solution it may be better to turn off the presolve.

Numerical Difficulties in the Simplex Optimizers

Though **MOSEK** is designed to minimize numerical instability, completely avoiding it is impossible when working in finite precision. **MOSEK** treats a "numerically unexpected behavior" event inside the optimizer as a *set-back*. The user can define how many set-backs the optimizer accepts; if that number is exceeded, the optimization will be aborted. Set-backs are a way to escape long sequences where the optimizer tries to recover from an unstable situation.

Examples of set-backs are: repeated singularities when factorizing the basis matrix, repeated loss of feasibility, degeneracy problems (no progress in objective) and other events indicating numerical difficulties. If the simplex optimizer encounters a lot of set-backs the problem is usually badly scaled; in such a situation try to reformulate it into a better scaled problem. Then, if a lot of set-backs still occur, trying one or more of the following suggestions may be worthwhile:

- Raise tolerances for allowed primal or dual feasibility: increase the value of
  - *dparam.basis_tol_x*, and
  - *dparam.basis_tol_s*.
- Raise or lower pivot tolerance: Change the *dparam.simplex_abs_tol_piv* parameter.
- Switch optimizer: Try another optimizer.
- Switch off crash: Set both *iparam.sim_primal_crash* and *iparam.sim_dual_crash* to 0.
- Experiment with other pricing strategies: Try different values for the parameters
  - *iparam.sim_primal_selection* and
  - *iparam.sim_dual_selection*.
- If you are using warm-starts, in rare cases switching off this feature may improve stability. This is controlled by the *iparam.sim_hotstart* parameter.
- Increase maximum number of set-backs allowed controlled by *iparam.sim_max_num_setbacks*.
- If the problem repeatedly becomes infeasible try switching off the special degeneracy handling. See the parameter *iparam.sim_degen* for details.

The Simplex Log

Below is a typical log output from the simplex optimizer:

```
Optimizer  - solved problem         : the primal
Optimizer  - Constraints            : 667
Optimizer  - Scalar variables       : 1424              conic                  : 0
Optimizer  - hotstart               : no
ITER      DEGITER(%)  PFEAS       DFEAS       POBJ                   DOBJ              ␣
↪     TIME      TOTTIME
0         0.00        1.43e+05    NA          6.5584140832e+03       NA                ␣
↪     0.00      0.02
1000      1.10        0.00e+00    NA          1.4588289726e+04       NA                ␣
↪     0.13      0.14
2000      0.75        0.00e+00    NA          7.3705564855e+03       NA                ␣
```

(continues on next page)

(continued from previous page)

```
↪    0.21      0.22
3000     0.67        0.00e+00    NA          6.0509727712e+03      NA          ␣
↪    0.29      0.31
4000     0.52        0.00e+00    NA          5.5771203906e+03      NA          ␣
↪    0.38      0.39
4533     0.49        0.00e+00    NA          5.5018458883e+03      NA          ␣
↪    0.42      0.44
```

The first lines summarize the problem the optimizer is solving. This is followed by the iteration log, with the following meaning:

- `ITER`: Number of iterations.
- `DEGITER(%)`: Ratio of degenerate iterations.
- `PFEAS`: Primal feasibility measure reported by the simplex optimizer. The numbers should be 0 if the problem is primal feasible (when the primal variant is used).
- `DFEAS`: Dual feasibility measure reported by the simplex optimizer. The number should be 0 if the problem is dual feasible (when the dual variant is used).
- `POBJ`: An estimate for the primal objective value (when the primal variant is used).
- `DOBJ`: An estimate for the dual objective value (when the dual variant is used).
- `TIME`: Time spent since this instance of the simplex optimizer was invoked (in seconds).
- `TOTTIME`: Time spent since optimization started (in seconds).

13.3 Conic Optimization - Interior-point optimizer

For conic optimization problems only an interior-point type optimizer is available. The same optimizer is used for quadratic optimization problems which are internally reformulated to conic form.

13.3.1 The homogeneous primal-dual problem

The interior-point optimizer is an implementation of the so-called homogeneous and self-dual algorithm. For a detailed description of the algorithm, please see [ART03]. In order to keep our discussion simple we will assume that **MOSEK** solves a conic optimization problem of the form:

$$\begin{array}{lll} \text{minimize} & c^T x & \\ \text{subject to} & Ax & = \quad b, \\ & x \in \mathcal{K} & \end{array} \tag{13.6}$$

where K is a convex cone. The corresponding dual problem is

$$\begin{array}{lll} \text{maximize} & b^T y & \\ \text{subject to} & A^T y + s & = \quad c, \\ & s \in \mathcal{K}^* & \end{array} \tag{13.7}$$

where $\mathcal{K}^*$ is the dual cone of $\mathcal{K}$. See Sec. 12.2 for definitions.

Since it is not known beforehand whether problem (13.6) has an optimal solution, is primal infeasible or is dual infeasible, the optimization algorithm must deal with all three situations. This is the reason that **MOSEK** solves the so-called homogeneous model

$$\begin{array}{rcl} Ax - b\tau & = & 0, \\ A^T y + s - c\tau & = & 0, \\ -c^T x + b^T y - \kappa & = & 0, \\ x & \in & \mathcal{K}, \\ s & \in & \mathcal{K}^*, \\ \tau, \kappa & \geq & 0, \end{array} \tag{13.8}$$

where y and s correspond to the dual variables in (13.6), and τ and κ are two additional scalar variables. Note that the homogeneous model (13.8) always has a solution since

$$(x, y, s, \tau, \kappa) = (0, 0, 0, 0, 0)$$

is a solution, although not a very interesting one. Any solution

$$(x^*, y^*, s^*, \tau^*, \kappa^*)$$

to the homogeneous model (13.8) satisfies

$$(x^*)^T s^* + \tau^* \kappa^* = 0$$

i.e. complementarity. Observe that $x^* \in \mathcal{K}$ and $s^* \in \mathcal{K}^*$ implies

$$(x^*)^T s^* \geq 0$$

and therefore

$$\tau^* \kappa^* = 0.$$

since $\tau^*, \kappa^* \geq 0$. Hence, at least one of τ^* and κ^* is zero.

First, assume that $\tau^* > 0$ and hence $\kappa^* = 0$. It follows that

$$\begin{aligned} A\frac{x^*}{\tau^*} &= b, \\ A^T\frac{y^*}{\tau^*} + \frac{s^*}{\tau^*} &= c, \\ -c^T\frac{x^*}{\tau^*} + b^T\frac{y^*}{\tau^*} &= 0, \\ x^*/\tau^* &\in \mathcal{K}, \\ s^*/\tau^* &\in \mathcal{K}^*. \end{aligned}$$

This shows that $\frac{x^*}{\tau^*}$ is a primal optimal solution and $(\frac{y^*}{\tau^*}, \frac{s^*}{\tau^*})$ is a dual optimal solution; this is reported as the optimal interior-point solution since

$$(x, y, s) = \left(\frac{x^*}{\tau^*}, \frac{y^*}{\tau^*}, \frac{s^*}{\tau^*}\right)$$

is a primal-dual optimal solution.

On other hand, if $\kappa^* > 0$ then

$$\begin{aligned} Ax^* &= 0, \\ A^T y^* + s^* &= 0, \\ -c^T x^* + b^T y^* &= \kappa^*, \\ x^* &\in \mathcal{K}, \\ s^* &\in \mathcal{K}^*. \end{aligned}$$

This implies that at least one of

$$c^T x^* < 0 \tag{13.9}$$

or

$$b^T y^* > 0 \tag{13.10}$$

holds. If (13.9) is satisfied, then x^* is a certificate of dual infeasibility, whereas if (13.10) holds then y^* is a certificate of primal infeasibility.

In summary, by computing an appropriate solution to the homogeneous model, all information required for a solution to the original problem is obtained. A solution to the homogeneous model can be computed using a primal-dual interior-point algorithm [And09].

13.3.2 Interior-point Termination Criterion

Since computations are performed in finite precision, and for efficiency reasons, it is not possible to solve the homogeneous model exactly in general. Hence, an exact optimal solution or an exact infeasibility certificate cannot be computed and a reasonable termination criterion has to be employed.

In every iteration k of the interior-point algorithm a trial solution

$$(x^k, y^k, s^k, \tau^k, \kappa^k)$$

to the homogeneous model is generated, where

$$x^k \in \mathcal{K}, s^k \in \mathcal{K}^*, \tau^k, \kappa^k > 0.$$

Therefore, it is possible to compute the values:

$$\begin{array}{rcl}
\rho_p^k & = & \arg\min_\rho \left\{\rho \mid \left\| A\frac{x^k}{\tau^k} - b \right\|_\infty \leq \rho\varepsilon_p(1 + \|b\|_\infty)\right\}, \\
\rho_d^k & = & \arg\min_\rho \left\{\rho \mid \left\| A^T\frac{y^k}{\tau^k} + \frac{s^k}{\tau^k} - c \right\|_\infty \leq \rho\varepsilon_d(1 + \|c\|_\infty)\right\}, \\
\rho_g^k & = & \arg\min_\rho \left\{\rho \mid \left(\frac{(x^k)^T s^k}{(\tau^k)^2}, |\frac{c^T x^k}{\tau^k} - \frac{b^T y^k}{\tau^k}|\right) \leq \rho\varepsilon_g \max\left(1, \frac{\min(|c^T x^k|, |b^T y^k|)}{\tau^k}\right)\right\}, \\
\rho_{pi}^k & = & \arg\min_\rho \left\{\rho \mid \left\| A^T y^k + s^k \right\|_\infty \leq \rho\varepsilon_i b^T y^k,\ b^T y^k > 0\right\} \text{ and} \\
\rho_{di}^k & = & \arg\min_\rho \left\{\rho \mid \left\| Ax^k \right\|_\infty \leq -\rho\varepsilon_i c^T x^k,\ c^T x^k < 0\right\}.
\end{array}$$

Note $\varepsilon_p, \varepsilon_d, \varepsilon_g$ and ε_i are nonnegative user specified tolerances.

Optimal Case

Observe ρ_p^k measures how far x^k/τ^k is from being a good approximate primal feasible solution. Indeed if $\rho_p^k \leq 1$, then

$$\left\| A\frac{x^k}{\tau^k} - b \right\|_\infty \leq \varepsilon_p(1 + \|b\|_\infty). \tag{13.11}$$

This shows the violations in the primal equality constraints for the solution x^k/τ^k is small compared to the size of b given ε_p is small.

Similarly, if $\rho_d^k \leq 1$, then $(y^k, s^k)/\tau^k$ is an approximate dual feasible solution. If in addition $\rho_g \leq 1$, then the solution $(x^k, y^k, s^k)/\tau^k$ is approximate optimal because the associated primal and dual objective values are almost identical.

In other words if $\max(\rho_p^k, \rho_d^k, \rho_g^k) \leq 1$, then

$$\frac{(x^k, y^k, s^k)}{\tau^k}$$

is an approximate optimal solution.

Dual Infeasibility Certificate

Next assume that $\rho_{di}^k \leq 1$ and hence

$$\left\| Ax^k \right\|_\infty \leq -\varepsilon_i c^T x^k \text{ and } -c^T x^k > 0$$

holds. Now in this case the problem is declared dual infeasible and x^k is reported as a certificate of dual infeasibility. The motivation for this stopping criterion is as follows. Let

$$\bar{x} := \frac{x^k}{-c^T x^k}$$

and it is easy to verify that

$$\|A\bar{x}\|_\infty \leq \varepsilon_i \text{ and } c^T\bar{x} = -1$$

which shows $\bar{x}$ is an approximate certificate of dual infeasibility, where ε_i controls the quality of the approximation.

Primal Infeasiblity Certificate

Next assume that $\rho_{pi}^k \leq 1$ and hence

$$\left\|A^T y^k + s^k\right\|_\infty \leq \varepsilon_i b^T y^k \text{ and } b^T y^k > 0$$

holds. Now in this case the problem is declared primal infeasible and (y^k, s^k) is reported as a certificate of primal infeasibility. The motivation for this stopping criterion is as follows. Let

$$\bar{y} := \frac{y^k}{b^T y^k} \text{ and } \bar{s} := \frac{s^k}{b^T y^k}$$

and it is easy to verify that

$$\left\|A^T \bar{y} + \bar{s}\right\|_\infty \leq \varepsilon_i \text{ and } b^T \bar{y} = 1$$

which shows (y^k, s^k) is an approximate certificate of dual infeasibility, where ε_i controls the quality of the approximation.

13.3.3 Adjusting optimality criteria

It is possible to adjust the tolerances ε_p, ε_d, ε_g and ε_i using parameters; see the next table for details. Note that although this section discusses the conic optimizer, if the problem was originally input as a quadratic or quadratically constrained optimization problem then the parameter names that apply are those from the third column (with infix `QO` instead of `CO`).

Table 13.2: Parameters employed in termination criterion

ToleranceParameter	Name (for conic problems)	Name (for quadratic problems)
ε_p	*dparam.intpnt_co_tol_pfeas*	*dparam.intpnt_qo_tol_pfeas*
ε_d	*dparam.intpnt_co_tol_dfeas*	*dparam.intpnt_qo_tol_dfeas*
ε_g	*dparam.intpnt_co_tol_rel_gap*	*dparam.intpnt_qo_tol_rel_gap*
ε_i	*dparam.intpnt_co_tol_infeas*	*dparam.intpnt_qo_tol_infeas*

The default values of the termination tolerances are chosen such that for a majority of problems appearing in practice it is not possible to achieve much better accuracy. Therefore, tightening the tolerances usually is not worthwhile. However, an inspection of (13.11) reveals that the quality of the solution depends on $\|b\|_\infty$ and $\|c\|_\infty$; the smaller the norms are, the better the solution accuracy.

The interior-point method as implemented by **MOSEK** will converge toward optimality and primal and dual feasibility at the same rate [And09]. This means that if the optimizer is stopped prematurely then it is very unlikely that either the primal or dual solution is feasible. Another consequence is that in most cases all the tolerances, ε_p, ε_d, ε_g and ε_i, have to be relaxed together to achieve an effect.

If the optimizer terminates without locating a solution that satisfies the termination criteria, for example because of a stall or other numerical issues, then it will check if the solution found up to that point satisfies the same criteria with all tolerances multiplied by the value of *dparam.intpnt_co_tol_near_rel*. If this is the case, the solution is still declared as optimal.

To conclude the discussion in this section, relaxing the termination criterion is usually not worthwhile.

13.3.4 The Interior-point Log

Below is a typical log output from the interior-point optimizer:

```
Optimizer  - threads                : 20
Optimizer  - solved problem         : the primal
Optimizer  - Constraints            : 1
Optimizer  - Cones                  : 2
Optimizer  - Scalar variables       : 6                 conic                  : 6
Optimizer  - Semi-definite variables: 0                 scalarized             : 0
Factor     - setup time             : 0.00              dense det. time        : 0.00
```

(continues on next page)

(continued from previous page)

```
Factor     - ML order time          : 0.00              GP order time          : 0.00
Factor     - nonzeros before factor : 1                 after factor           : 1
Factor     - dense dim.             : 0                 flops                  : 1.
↪70e+01
ITE PFEAS    DFEAS    GFEAS    PRSTATUS   POBJ              DOBJ              MU
↪ TIME
0   1.0e+00  2.9e-01  3.4e+00  0.00e+00   2.414213562e+00   0.000000000e+00   1.0e+00
↪ 0.01
1   2.7e-01  7.9e-02  2.2e+00  8.83e-01   6.969257574e-01   -9.685901771e-03  2.7e-01
↪ 0.01
2   6.5e-02  1.9e-02  1.2e+00  1.16e+00   7.606090061e-01   6.046141322e-01   6.5e-02
↪ 0.01
3   1.7e-03  5.0e-04  2.2e-01  1.12e+00   7.084385672e-01   7.045122560e-01   1.7e-03
↪ 0.01
4   1.4e-08  4.2e-09  4.9e-08  1.00e+00   7.071067941e-01   7.071067599e-01   1.4e-08
↪ 0.01
```

The first line displays the number of threads used by the optimizer and the second line indicates if the optimizer chose to solve the primal or dual problem (see *iparam.intpnt_solve_form*). The next lines display the problem dimensions as seen by the optimizer, and the `Factor...` lines show various statistics. This is followed by the iteration log.

Using the same notation as in Sec. 13.3.1 the columns of the iteration log have the following meaning:

- `ITE`: Iteration index k.
- `PFEAS`: $\left\|Ax^k - b\tau^k\right\|_\infty$. The numbers in this column should converge monotonically towards zero but may stall at low level due to rounding errors.
- `DFEAS`: $\left\|A^T y^k + s^k - c\tau^k\right\|_\infty$. The numbers in this column should converge monotonically towards zero but may stall at low level due to rounding errors.
- `GFEAS`: $|-c^T x^k + b^T y^k - \kappa^k|$. The numbers in this column should converge monotonically towards zero but may stall at low level due to rounding errors.
- `PRSTATUS`: This number converges to 1 if the problem has an optimal solution whereas it converges to -1 if that is not the case.
- `POBJ`: $c^T x^k/\tau^k$. An estimate for the primal objective value.
- `DOBJ`: $b^T y^k/\tau^k$. An estimate for the dual objective value.
- `MU`: $\frac{(x^k)^T s^k + \tau^k \kappa^k}{n+1}$. The numbers in this column should always converge to zero.
- `TIME`: Time spent since the optimization started (in seconds).

13.4 The Optimizer for Mixed-Integer Problems

Solving optimization problems where one or more of the variables are constrained to be integer valued is called Mixed-Integer Optimization (MIO). For an introduction to model building with integer variables, the reader is recommended to consult the **MOSEK** Modeling Cookbook, and for further reading we highlight textbooks such as [Wol98] or [CCornuejolsZ14].

MOSEK can perform mixed-integer

- linear (MILO),
- quadratic (MIQO) and quadratically constrained (MIQCQO), and
- conic (MICO)

optimization, except for mixed-integer semidefinite problems.

By default the mixed-integer optimizer is run-to-run deterministic. This means that if a problem is solved twice on the same computer with identical parameter settings and no time limit, then the obtained solutions will be identical. The mixed-integer optimizer is parallelized, i.e., it can exploit multiple cores during the optimization.

In practice, it often happens that the integer variables in MIO problems are actuall binary variables, taking values in $\{0, 1\}$, leading to Mixed- or pure binary problems. In the general setting however, an integer variable may have arbitrary lower and upper bounds.

13.4.1 Branch-and-Bound

In order to succeed in solving mixed-integer problems, it can be useful to have a basic understanding of the underlying solution algorithms. The most important concept in this regard is arguably the so-called Branch-and-Bound algorithm, employed also by **MOSEK**. The more experienced reader may skip this section and advance directly to Sec. 13.4.2.

In order to comprehend Branch-and-Bound, the concept of a *relaxation* is important. Consider for example a mixed-integer linear optimization problem of minimization type

$$\begin{array}{rcllll} z^* & = & \text{minimize} & c^T x & & \\ & & \text{subject to} & Ax & = & b \\ & & & x \geq 0 & & \\ & & & x_j \in \mathbb{Z}, & \forall j \in \mathcal{J}. & \end{array} \tag{13.12}$$

It has the continuous relaxation

$$\begin{array}{rcllll} \underline{z} & = & \text{minimize} & c^T x & & \\ & & \text{subject to} & Ax & = & b \\ & & & x \geq 0, & & \end{array} \tag{13.13}$$

simply obtained by ignoring the integrality restrictions. The first step in Branch-and-Bound is to solve this so-called *root* relaxation, which is a continuous optimization problem. Since (13.13) is less constrained than (13.12), one certainly gets

$$\underline{z} \leq z^*,$$

and $\underline{z}$ is therefore called the *objective bound*: it bounds the optimal objective value from below.

After the solution of the root relaxation, in the most likely outcome there will be one or more integer constrained variables with fractional values, i.e., violating the integrality constraints. Branch-and-Bound now takes such a variable, $x_j = f_j \in \mathbb{R} \backslash \mathbb{Z}$ with $j \in \mathcal{J}$, say, and creates two branches leading to relaxations with the additional constraint $x_j \leq \lfloor f_j \rfloor$ or $x_j \geq \lceil f_j \rceil$, respectively. The intuitive idea here is to exclude the undesired fractional value from the outcomes in the two created branches. If the integer variable was actually a binary variable, branching would lead to fixing its value to 0 in one branch, and to 1 in the other.

The Branch-and-Bound process continues in this way and successively solves relaxations and creates branches to refined relaxations. Whenever the solution $\hat{x}$ to some relaxation does not violate any integrality constraints, it is feasible to (13.12) and is called an *integer feasible solution*. There is no guarantee though that it is also optimal, its solution value $\bar{z} := c^T \hat{x}$ is only an upper bound on the optimal objective value,

$$z^* \leq \bar{z}.$$

By the successive addition of constraints in the created branches, the objective bound $\underline{z}$ (now defined as the minimum over all solution values of so far solved relaxations) can only increase during the algorithm. At the same time, the upper bound $\bar{z}$ (the solution value of the best integer feasible solution encountered so far, also called *incumbent solution*) can only decrease during the algorithm. Since at any time we also have

$$\underline{z} \leq z^* \leq \bar{z},$$

objective bound and incumbent solution value are encapsulating the optimal objective value, eventually converging to it.

The Branch-and-Bound scheme can be depicted by means of a tree, where branches and relaxations correspond to edges and nodes. Figure Fig. 13.1 shows an example of such a tree. The strength of Branch-and-Bound is its ability to prune nodes in this tree, meaning that no new child nodes will be created. Pruning can occur in several cases:

- A relaxation leads to an integer feasible solution $\hat{x}$. In this case we may update the incumbent and its solution value $\bar{z}$, but no new branches need to be created.
- A relaxation is infeasible. The subtree rooted at this node cannot contain any feasible relaxation, so it can be discarded.
- A relaxation has a solution value that exceeds $\bar{z}$. The subtree rooted at this node cannot contain any integer feasible solution with a solution value better than the incumbent we already have, so it can be discarded.

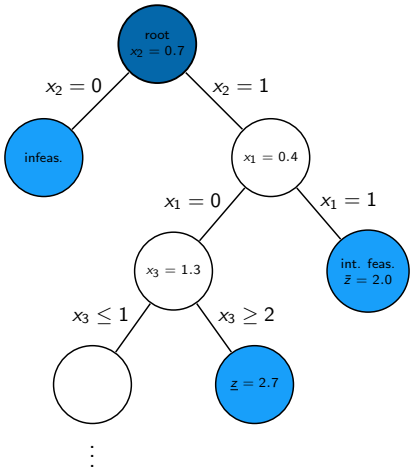

Fig. 13.1: An examplary Branch-and-Bound tree. Pruned nodes are shown in light blue.

Having objective bound and incumbent solution value is a quite fundamental property of Branch-and-Bound, and helps to asses solution quality and control termination of the algorithm, as we detail in the next section. Note that the above explanation is coined for minimization problems, but the Branch-and-bound scheme has a straightforward extension to maximization problems.

13.4.2 Solution quality and termination criteria

The issue of terminating the mixed-integer optimizer is rather delicate. Mixed-integer optimization is generally much harder than continuous optimization; in fact, solving continuous sub-problems is just one component of a mixed-integer optimizer. Despite the ability to prune nodes in the tree, the computational effort required to solve mixed-integer problems grows exponentially with the size of the problem in a worst-case scenario (solving mixed-integer problems is NP-hard). For instance, a problem with n binary variables, may require the solution of 2^n relaxations. The value of 2^n is huge even for moderate values of n. In practice it is often advisable to accept near-optimal or appproximate solutions in order to counteract this complexity burden. The user has numerous possibilities of influencing optimizer termination with various parameters, in particular related to solution quality, and the most important ones are highlighted here.

Solution quality in terms of optimality

In order to assess the quality of any incumbent solution in terms of its objective value, one may check the *optimality gap*, defined as

$$\epsilon = |(\text{incumbent solution value}) - (\text{objective bound})| = |\bar{z} - \underline{z}|.$$

It measures how much the objectives of the incumbent and the optimal solution can deviate in the worst case. Often it is more meaningful to look at the *relative optimality gap*

$$\epsilon_{\text{rel}} = \frac{|\bar{z} - \underline{z}|}{\max(\delta_1, |\bar{z}|)}.$$

This is essentially the above *absolute* optimality gap normalized against the magnitude of the incumbent solution value; the purpose of the (small) constant δ_1 is to avoid overweighing incumbent solution values that are very close to zero. The relative optimality gap can thus be interpreted as answering the question: *"Within what fraction of the optimal solution is the incumbent solution in the worst case?"*

Absolute and relative optimality gaps provide useful means to define termination criteria for the mixed-integer optimizer in **MOSEK**. The idea is to terminate the optimization process as soon as the quality of the incumbent solution, measured in absolute or relative gap, is good enough. In fact, whenever an incumbent solution is located, the criterion

$$\epsilon \leq \delta_2 \text{ or } \epsilon_{\text{rel}} \leq \delta_3$$

is checked. If satisfied, i.e., if either absolute or relative optimality gap are below the thresholds δ_2 or δ_3 (see Table 13.3), the optimizer terminates and reports the incumbent as an optimal solution. The optimality gaps at termination can always be retrieved through the information items *dinfitem.mio_obj_abs_gap* and *dinfitem.mio_obj_rel_gap*.

The tolerances discussed above can be adjusted using suitable parameters, see Table 13.3. By default, the optimality parameters δ_2 and δ_3 are quite small, i.e., restrictive. These default values for the absolute and relative gap amount to solving any instance to (almost) optimality: the incumbent is required to be within at most a tiny percentage of the optimal solution. As anticipated, this is not tractable in many practical situations, and one should resort to finding near-optimal solutions quickly rather than insisting on finding the optimal one. It may happen, for example, that an optimal or close-to-optimal solution is found very early by the optimizer, but it spends a huge amount of further computational time for branching, trying to increase $\underline{z}$ that last missing bit: a typical situation that practioneers would want to avoid. The concept of optimality gaps is fundamental for controlling solution quality when resorting to near-optimal solutions.

MIO performance tweaks: termination criteria

One of the first things to do in order to cut down excessive solution time is to increase the relative gap tolerance *dparam.mio_tol_rel_gap* to some non-default value, so as to not insist on finding optimal solutions. Typical values could be $0.01, 0.05$ or 0.1, guaranteeing that the delivered solutions lie within $1\%, 5\%$ or 10% of the optimum. Increasing the tolerance will lead to less computational time spent by the optimizer.

Solution quality in terms of feasibility

For an optimizer relying on floating-point arithmetic like the mixed-integer optimizer in **MOSEK**, it may be hard to achieve exact integrality of the solution values of integer variables in most cases, and it makes sense to numerically relax this constraint. Any candidate solution $\hat{x}$ is accepted as integer feasible if the criterion

$$\min(\hat{x_j} - \lfloor \hat{x_j} \rfloor, \lceil \hat{x_j} \rceil - \hat{x_j}) \leq \delta_4 \quad \forall j \in \mathcal{J}$$

is satisfied, meaning that $\hat{x_j}$ is at most δ_4 away from the nearest integer. As above, δ_4 can be adjusted using a parameter, see Table 13.3, and impacts the quality of the acieved solution in terms of integer feasibility. By influencing what solution may be accepted as imcumbent, it can also have an impact on the termination of the optimizer.

MIO performance tweaks: feasibility criteria

Whether increasing the integer feasibility tolerance *dparam.mio_tol_abs_relax_int* leads to less solution time is highly problem dependent. Intuitively, the optimizer is more flexible in finding new incumbent soutions so as to improve $\bar{z}$. But this effect has do be examined with care on indivuidual instances: it may worsen solution quality with no effect at all on the solution time. It may in some cases even lead to contrary effects on the solution time.

Table 13.3: Tolerances for the mixed-integer optimizer.

Tolerance	Parameter name	Default value
δ_1	*dparam.mio_rel_gap_const*	1.0e-10
δ_2	*dparam.mio_tol_abs_gap*	0.0
δ_3	*dparam.mio_tol_rel_gap*	1.0e-4
δ_4	*dparam.mio_tol_abs_relax_int*	1.0e-5

Further controlling optimizer termination

There are more ways to limit the computational effort employed by the mixed-integer optimizer by simply limiting the number of explored branches, solved relaxations or updates of the incumbent solution. When any of the imposed limits is hit, the optimizer terminates and the incumbent solution may be retrieved. See Table 13.4 for a list of corresponding parameters. In contrast to the parameters discussed in Sec. 13.4.2, interfering with these does not maintain any guarantees in terms of solution quality.

Table 13.4: Other parameters affecting the integer optimizer termination criterion.

Parameter name	Explanation
iparam.mio_max_num_branches	Maximum number of branches allowed.
iparam.mio_max_num_relaxs	Maximum number of relaxations allowed.
iparam.mio_max_num_solutions	Maximum number of feasible integer solutions allowed.

13.4.3 The Mixed-Integer Log

The Branch-and-Bound scheme from Sec. 13.4.1 is only the basic skeleton of the mixed-integer optimizer in **MOSEK**, and several components are built on top of that in order to enhance its functionality and increase its speed. A mixed-integer optimizer is sometimes referred to as a *"giant bag of tricks"*, and it would be impossible to describe all of these tricks here. Yet, some of the additional components are worth mentioning. They can be influenced by various user parameters, and although the default values of these parameters are optimized to work well on average mixed-integer problems, it may pay off to adjust them for an individual problem, or a specific problem class. The mixed-integer log can give insights on which parameters might be worth an adjustment. Below is a typical log output:

```
Presolve started.
Presolve terminated. Time = 0.23, probing time =  0.09
Presolved problem: 1176 variables, 1344 constraints, 4968 non-zeros
Presolved problem: 328 general integer, 392 binary, 456 continuous
Clique table size: 55
Symmetry factor  : 0.79 (detection time = 0.01)
Removed blocks   : 2
BRANCHES RELAXS   ACT_NDS  DEPTH    BEST_INT_OBJ         BEST_RELAX_OBJ       REL_GAP(
↪%)  TIME
```

(continues on next page)

(continued from previous page)

```
0        0        1        0        8.3888091139e+07     NA                   NA       ␣
↪    0.2
0        1        1        0        8.3888091139e+07     2.5492512136e+07     69.61    ␣
↪    0.3
0        1        1        0        3.1273162420e+07     2.5492512136e+07     18.48    ␣
↪    0.4
0        1        1        0        2.6047699632e+07     2.5492512136e+07     2.13     ␣
↪    0.4
Rooot cut generation started.
0        1        1        0        2.6047699632e+07     2.5492512136e+07     2.13     ␣
↪    0.4
0        2        1        0        2.6047699632e+07     2.5589986247e+07     1.76     ␣
↪    0.4
Rooot cut generation terminated. Time = 0.05
0        4        1        0        2.5990071367e+07     2.5662741991e+07     1.26     ␣
↪    0.5
0        8        1        0        2.5971002767e+07     2.5662741991e+07     1.19     ␣
↪    0.6
0        11       1        0        2.5925040617e+07     2.5662741991e+07     1.01     ␣
↪    0.6
0        12       1        0        2.5915504014e+07     2.5662741991e+07     0.98     ␣
↪    0.6
2        23       1        0        2.5915504014e+07     2.5662741991e+07     0.98     ␣
↪    0.7
14       35       1        0        2.5915504014e+07     2.5662741991e+07     0.98     ␣
↪    0.7

[ ... ]

Objective of best integer solution : 2.578282162804e+07
Best objective bound               : 2.569877601306e+07
Construct solution objective       : Not employed
User objective cut value           : Not employed
Number of cuts generated           : 192
  Number of Gomory cuts            : 52
  Number of CMIR cuts              : 137
  Number of clique cuts            : 3
Number of branches                 : 29252
Number of relaxations solved       : 31280
Number of interior point iterations: 16
Number of simplex iterations       : 105440
Time spend presolving the root     : 0.23
Time spend optimizing the root     : 0.07
Mixed integer optimizer terminated. Time: 6.96
```

The main part here is the iteration log, a progressing series of similar rows reflecting the progress made during the Branch-and-bound process. The columns have the following meanings:

- `BRANCHES`: Number of branches / nodes generated.
- `RELAXS`: Number of relaxations solved.
- `ACT_NDS`: Number of active / non-processed nodes.
- `DEPTH`: Depth of the last solved node.
- `BEST_INT_OBJ`: The incumbent solution / best integer objective value, $\bar{z}$.
- `BEST_RELAX_OBJ`: The objective bound, $\underline{z}$.

- `REL_GAP(%)`: Relative optimality gap, $100\% \cdot \epsilon_{\text{rel}}$
- `TIME`: Time (in seconds) from the start of optimization.

Also a short solution summary with several statistics is printed. When the solution time for a mixed-integer problem has to be cut down, the log can help to understand where time is spent and what might be improved. We go into some more detail about some further items in the mixed-integer log giving hints about individual components of the optimizer. Alternatively, most of these items can also be retrieved as information items, see Sec. 7.6.

Presolve

Similar to the case of continuous problems, see Sec. 13.1, the mixed-integer optimizer applies various presolve reductions before the actual Branch-and-bound is initiated. The first lines of the mixed-integer log contain a summary of the presolve process, including the time spent therein (`Presolve terminated. Time = 0.23...`). Just as in the continuous case, the use of presolve can be controlled with the parameter *iparam.presolve_use*. If presolve time seems excessive, instead of switching it off completely one may also try to reduce the time spent in one or more of its individual components. On some models it can also make sense to increase the use of a certain presolve technique. Table Table 13.5 lists some of these with their respective parameters.

Table 13.5: Parameters affecting presolve

Parameter name	Explanation	Possible reference in log
iparam.mio_probing_level	Probing aggressivity level.	`... probing time = 0.09`
iparam.mio_symmetry_level	Symmetry detection aggressivity level.	`Symmetry factor : 0.79 (detection time = 0.01)`
iparam.mio_independent_block_le	Block structure detection level, see Sec. 13.4.3.	`Removed blocks : 2`
dparam.mio_clique_table_size_fa	Maximum size of the clqiue table.	`Clique table size: 55`
*iparam.mio_presolve_aggregator_*	Should variable agggregation be enabled?	–

Primal Heuristics

It might happen that the value in the colum `BEST_INT_OBJ` stalls over a long period of log lines, an indication that the optimizer has a hard time improving the incumbent solution, i.e., $\bar{z}$. Solving relaxations in the tree to an integer feasible solution $\hat{x}$ is not the only way to find new incumbent solutions. There is a variety of procedures that, given a mixed-integer problem in a generic form like (13.12), attempt to produce integer feasible solutions in an ad-hoc way. These procedures are called Primal Heuristics, and several of them are implemented in **MOSEK**. For example, whenever a relaxation leads to a fractional solution, one may round the solution values of the integer variables, in various ways, and hope that the outcome is still feasible to the remaining constraints. Primal heuristics are mostly employed while processing the root node, but play a role throughout the whole solution process. The goal of a primal heuristic is to improve the incumbent solution and thus the bound $\bar{z}$, and this can of course affect the quality of the solution that is returned after termination of the optimizer. The user parameters affecting primal heuristics are listed in Table 13.6.

MIO performance tweaks: primal heuristics

- If the mixed-integer optimizer struggles to improve the incumbent solution `BEST_INT_OBJ`, it can be helpful to intensify the use of primal heuristics.
  - Set parameters related to primal heuristics to more aggressive values than the default ones, so that more effort is spent in this component. A List of the respective parameters can be found in Table 13.6. In particular, if the optimizer has difficulties finding any integer feasible solution at all, indicated by `NA` in the column `BEST_INT_OBJ` in the mixed-integer log, one may try to activate a construction heuristic like the Feasibility Pump with *iparam.mio_feaspump_level*.

  - Specify a good initial solution: In many cases a good feasible solution is either known or easily computed using problem-specific knowledge that the optimizer does not have. If so, it is usually worthwhile to use this as a starting point for the mixed-integer optimizer. See also the parameter *iparam.mio_construct_sol*, and Section Sec. 6.8.2.
  - For feasibility problems, i.e., problems having a constant objective, the goal is to find a single integer feasible solution, and this can be hard by itself on some instances. Try setting the objective to something meaningful anyway, even if the underlying application does not require this. After all, the feasible set is not changed, but the optimizer might benefit from being able to pursue a concrete goal.

- In rare cases it may also happen that the optimizer spends an excessive amount of time on primal heuristics without drawing any benefit from it, and one may try to limit their use with the respective parameters.

Table 13.6: Parameters affecting primal heuristics

Parameter name	Explanation
iparam.mio_heuristic_level	Primal heuristics aggressivity level.
iparam.mio_rins_max_nodes	Maximum number of nodes allowed in the RINS heuristic.
iparam.mio_rens_max_nodes	Maximum number of nodes allowed in the RENS heuristic.
iparam.mio_crossover_max_nodes	Maximum number of nodes allowed in the Crossover heuristic.
iparam.mio_opt_face_max_nodes	Maximum number of nodes allowed in the optimal face heuristic.
iparam.mio_feaspump_level	Way of using the Feasibility Pump heuristic.

Cutting Planes

It might as well happen that the value in the colum `BEST_RELAX_OBJ` stalls over a long period of log lines, an indication that the optimizer has a struggles to improve the objective bound $\underline{z}$. A component of the optimizer designed to act on the objective bound is given by Cutting planes, also called cuts or valid inequalities. Cuts do not remove any integer feasible solutions from the feasible set of the mixed-integer problem (13.12). They may, however, remove solutions from the feasible set of the relaxation (13.13), ideally making it a *stronger* relaxation with better objective bound.

As an example, take the constraints

$$2x_1 + 3x_2 + x_3 \leq 4, \quad x_1, x_2 \in \{0, 1\}, \quad x_3 \geq 0. \tag{13.14}$$

One may realize that there cannot be a feasible solution in which both binary variables take on a value of 1. So certainly

$$x_1 + x_2 \leq 1 \tag{13.15}$$

is a valid inequality (there is no integer solution satisfying (13.14), but violating (13.15)). The latter does cut off a portion of the feasible region of the continuous relaxation of (13.14) though, obtained by replacing $x_1, x_2 \in \{0, 1\}$ with $x_1, x_2 \in [0, 1]$. For example, the fractional point $(x_1, x_2, x_3) = (0.5, 1, 0)$ is feasible to the relaxation, but violates the cut (13.15).

There are many classes of general-purpose cuttting planes that may be generated for a mixed-integer problem in a generic form like (13.12), and **MOSEK**'s mixed-integer optimizer supports several of them. For instance, the above is an example of a so-called clique cut. The most effort on generating cutting planes is spent after the solution of the root relaxation; the beginning and the end of root cut generation is highlighted in the log, and the number of log lines in between reflects to the computational effort spent here. Also the solution summary at the end of the log highlights for each cut class the number of generated cuts. Cuts can also be generated later on in the tree, which is why we also use the term Branch-and-cut, an extension of the basic Branch-and-bound scheme. Cuts aim at improving the objective bound $\underline{z}$ and can thus have significant impact on the solution time. The user parameters affecting cut generation can be seen in Table 13.7.

MIO performance tweaks: cutting planes

- If the mixed-integer optimizer struggles to improve the objective bound `BEST_RELAX_OBJ`, it can be helpful to intensify the use of cutting planes.
  - Some types of cutting planes are not activated by default, but doing so may help to improve the objective bound.
  - The parameters *dparam.mio_tol_rel_dual_bound_improvement* and *iparam.mio_cut_selection_level* determine how aggressively cuts will be generated and selected.
  - If some valid inequalities can be deduced from problem-specific knowledge that the optimizer does not have, it may be helpful to add these to the problem formulation as constraints. This has to be done with care, since there is a tradeoff between the benefit obtained from an improved objective boud, and the amount of additional constraints that make the relaxations larger.
- In rare cases it may also be observed that the optimizer spends an excessive effort on cutting planes, and one may limit their use with *iparam.mio_max_num_root_cut_rounds*, or by disabling a certain type of cutting planes.

Table 13.7: Parameters affecting cutting planes

Parameter name	Explanation
iparam.mio_cut_clique	Should clique cuts be enabled?
iparam.mio_cut_cmir	Should mixed-integer rounding cuts be enabled?
iparam.mio_cut_gmi	Should GMI cuts be enabled?
iparam.mio_cut_implied_bound	Should implied bound cuts be enabled?
iparam.mio_cut_knapsack_cover	Should knapsack cover cuts be enabled?
iparam.mio_cut_lipro	Should lift-and-project cuts be enabled?
iparam.mio_cut_selection_level	Cut selection aggressivity level.
iparam.mio_max_num_root_cut_round	Maximum number of root cut rounds.
dparam.mio_tol_rel_dual_bound_imp	Minimum required objective bound improvement during root cut generation.

Restarts

The mixed-integer optimizer employs so-called restarts, i.e., if the progress made while exploring the tree is deemed unsufficient, it might decide to restart the solution process from scratch, possibly making use of the information collected so far. When a restart happens, this is displayed in the log:

```
[ ... ]

1948     4664     699      36       NA                   1.1800000000e+02     NA      ␣
↪    7.2
1970     4693     705      50       NA                   1.1800000000e+02     NA      ␣
↪    7.2

Performed MIP restart 1.
Presolve started.
Presolve terminated. Time = 0.01, probing time =  0.00
Presolved problem: 523 variables, 765 constraints, 3390 non-zeros
Presolved problem: 0 general integer, 404 binary, 119 continuous
Clique table size: 143
BRANCHES RELAXS   ACT_NDS  DEPTH    BEST_INT_OBJ         BEST_RELAX_OBJ       REL_GAP(
↪%)  TIME
1988     4729     1        0        NA                   1.1800000000e+02     NA      ␣
```

(continues on next page)

(continued from previous page)

```
↪    7.3
1988     4730     1        0        4.0000000000e+01     1.1800000000e+02     195.00 ␣
↪    7.3

[ ... ]
```

Restarts tend to be useful especially for hard models. However, in individual cases the optimizer may decide to perform a restart while it would have been better to continue exploring the tree. Their use can be controlled with the parameter *iparam.mio_max_num_restarts*.

Block decomposition

Sometimes the optimizer faces a model that actually represents two or more completely independent subproblems. For a linear problem such as (13.13), this means that the constraint matrix A is a block-diagonal. Block-diagonal structure can occur after **MOSEK** applies some presolve reductions, e.g., a variable is fixed that was the only variable connecting two otherwise independent subproblems. Or, more rarely, the original model provided by the user is already block-diagonal.

In principle, solving such blocks independently is easier than letting the optimizer work on the single, large model, and **MOSEK** thus tries to exploit this structure. Some blocks may be completely solved and removed from the model during presolve, which can be seen by a line at the end of the presolve summary, see also Sec. 13.4.3. If after presolve there are still independent blocks, **MOSEK** can apply a dedicated algorithm to solve them independently while periodically combining their individual solution statusses (such as incumbent solutions and objective bounds) to the solution status of the original model. Just like the removal of blocks during presolve, the application of this latter strategy is indicated in the log:

```
[ ... ]

15       38       1        0        4.1759800000e+05     3.8354200000e+05     8.16  ␣
↪    0.9
Root cut generation started.
15       38       1        0        4.1759800000e+05     3.8354200000e+05     8.16  ␣
↪    1.1
Root cut generation terminated. Time = 0.11
15       40       1        0        4.1645600000e+05     3.8934425000e+05     6.51  ␣
↪    2.0
15       41       1        0        4.1622400000e+05     3.8934425000e+05     6.46  ␣
↪    2.0
23       52       1        0        4.1622400000e+05     3.8934425000e+05     6.46  ␣
↪    2.0
Decomposition solver started with 5 independent blocks.
532      425      5        118      4.1592600000e+05     3.8935275000e+05     6.39  ␣
↪    4.5
1858     11911    815      286      4.1007800000e+05     3.8946400000e+05     5.03  ␣
↪    11.8

[ ... ]
```

How block-diagonal structure is detected and handled by the optimizer can be controlled with the parameter *iparam.mio_independent_block_level*.

13.4.4 Mixed-Integer Nonlinear Optimization

Due to the involved non-linearities, MI(QC)QO or MICO problems are on average harder than MILO problems of comparable size. Yet, the Branch-and-Bound scheme can be applied to these probelm classes in a straightforward manner. The relaxations have to be solved as conic problems with the interior point algorithm in that case, see Sec. 13.3, opposed to MILO where it is often beneficial to solve relaxations with the dual simplex method, see Sec. 13.2.3. There is another solution approach for these types of problems implemented in **MOSEK**, namely the Outer-Approximation algorithm, making use of dynamically refined linear approximations of the non-linearities.

MICO performance tweaks: choice of algorithm

Whether conic Branch-and-Bound or Outer-Approximation is applied to a mixed-integer conic problem can be set with *iparam.mio_conic_outer_approximation*. The best value for this option is highly problem dependent.

MI(QC)QO

MOSEK is specialized in solving linear and conic optimization problems, both with or without mixed-integer variables. Just like for continuous problems, mixed-integer quadratic problems are converted internally to conic form, see Sec. 12.4.1

Contrary to the continuous case, **MOSEK** can solve certain mixed-integer quadratic problems where one or more of the involved matrices are not positive semidefinite, so-called non-convex MI(QC)QO problems. These are automatically reformulated to an equivalent convex MI(QC)QO problem, provided that such a reformulation is possible on the given instance (otherwiese **MOSEK** will reject the problem and issue an error message). The concept of reformulations can also affect the solution times of MI(QC)QO problems.

MI(QC)QO performance tweaks: applying a reformulation method

There are several reformulation methods for MI(QC)QO problems, available through the parameter *iparam.mio_qcqo_reformulation_method*. The chosen method can have significant impact on the mixed-integer optimizer's speed on such problems, both convex and non-convex. The best value for this option is highly problem dependent.

13.4.5 Disjunctive constraints

Problems with disjunctive constraints (DJC) see Sec. 6.9 are typically reformulated to mixed-integer problems, and even if this is not the case they are solved with an algorithm that is based on the mixed-integer optimizer. In **MOSEK**, these problems thus fall into the realm of MIO. In particular, **MOSEK** automatically attempts to replace any DJC by so called big-M constraints, potentially after transforming it to several, less complicated DJCs. As an example, take the DJC

$$[z = 0] \vee [z = 1, x_1 + x_2 \geq 1000],$$

where $z \in \{0, 1\}$ and $x_1, x_2 \in [0, 750]$. This is an example of a DJC formulation of a so-called indicator constraint. A big-M reformulation is given by

$$x_1 + x_2 \geq 1000 - M \cdot (1 - z),$$

where $M > 0$ is a large constant. The practical difficulty of these constructs is that M should always be sufficiently large, but ideally not larger. Too large values for M can be harmful for the mixed-integer optimizer. During presolve, and taking into account the bounds of the involved variables, **MOSEK** automatically reformulates DJCs to big-M constraints if the required M values do not exceed the parameter *dparam.mio_djc_max_bigm*. From a performance point-of-view, all DJCs would ideally be linearized to big-Ms after presolve without changing this parameter's default value of 1.0e6. Whether or not this is the case can be seen by retrieving the information item *iinfitem.mio_presolved_numdjc*, or by a line in the mixed-integer optimizer's log as in the example below. Both state the number of remaining disjunctions after presolve.

```
Presolved problem: 305 variables, 204 constraints, 708 non-zeros
Presolved problem: 0 general integer, 100 binary, 205 continuous
Presolved problem: 100 disjunctions
Clique table size: 0
BRANCHES RELAXS   ACT_NDS  DEPTH    BEST_INT_OBJ         BEST_RELAX_OBJ       REL_GAP(
↪%)  TIME
0        1        1        0        NA                   0.0000000000e+00     NA      ␣
↪    0.0
0        1        1        0        5.0574653969e+05     0.0000000000e+00     100.00  ␣
↪    0.0

[ ... ]
```

DJC performance tweaks: managing variable bounds

- Always specify the tightest known bounds on the variables of any problem with DJCs, even if they seem trivial from the user-perspective. The mixed-integer optimizer can only benefit from these when reformulating DJCs and thus gain performance; even if bounds don't help with reformulations, it is very unlikely that they hurt the optimizer.
- Increasing *dparam.mio_djc_max_bigm* can lead to more DJC reformulations and thus increase optimizer speed, but it may in turn hurt numerical solution quality and has to be examined with care. The other way round, on numerically challenging instances with DJCs, decreasing *dparam.mio_djc_max_bigm* may lead to numerically more robust solutions.

13.4.6 Randomization

A mixed-integer optimizer is usually prone to performance variability, meaning that a small change in either

- problem data, or
- computer hardware, or
- algorithmic parameters

can lead to significant changes in solution time, due to different solution paths in the Branch-and-cut tree. In extreme cases the exact same problem can vary from being solvable in less than a second to seemingly unsolvable in any reasonable amount of time on a different computer.

One practical implication of this is that one should ideally verify whether a seemingly beneficial set of parameters, established experimentally on a single problem, is still beneficial (on average) on a larger set of problems from the same problem class. This protects against making parameter changes that had positive effects only due to random effects on that single problem.

In the absence of a large set of test problems, one may also change the random seed of the optimizer to a series of different values in order to hedge against drawing such wrong conclusions regarding parameters. The random seed, accessible through *iparam.mio_seed*, impacts for example random tie-breaking in many of the mixed-integer optimizer's components. Changing the random seed can be combined with a permutation of the problem data to further incite randomness, accessible through the parameter *iparam.mio_data_permutation_method*.

13.4.7 Further performance tweaks

In addition to what was mentioned previously, there may be other ways to speed up the solution of a given mixed-integer problem. For example, there are further user parameters affecting some algorithmic settings in the mixed-integer optimizer. As mentioned above, default parameter values are optimized to work well on average, but on individual problems they may be adjusted.

MIO performance tweaks: miscellaneous

- While exploring the tree, the optimizer applies certain strategies to decide which fractional variable to branch on, see Sec. 13.4.1. The chosen strategy can have a big impact on performance, and may be controlled with *iparam.mio_var_selection*.
- Similarly, the strategy to chose the next node to explore in the tree is controlled with *iparam.mio_node_selection*.
- The optimizer employs specialized techniques to learn from infeasible nodes and use that knowledge to avoid creating similar nodes in other parts of the tree. The effort spent here can be influenced with *iparam.mio_dual_ray_analysis_level* and *iparam.mio_conflict_analysis_level*.
- When relaxations in the tree are linear optimization problems (e.g., in MILO or when solving MICO probelms with the Outer-Approximation method), it is usually best to employ the dual simplex method for their solution. In rare cases the primal simplex method may actually be the better choice, and this can be set with the parameter *iparam.mio_node_optimizer*.
- Some problems are numerically more challenging than others, for example if the ratio between the smallest and the largest involved coefficients is large, say $\geq 1e9$. An indication of numerical issues are, for example, large violations in the final solution, observable in the solution summery of the log output, see Sec. 8.1.3. Similarly, a problem that is known to be feasible by the user may be declared infeasible by the optimizer. In such cases it is usually best to try to rescale the model. Otherwise, the mixed-integer optimizer can be instructed to be more cautios regarding numerics with the parameter *iparam.mio_numerical_emphasis_level*. This may in turn be at the cost of solution speed though.
- Improve the formulation: A MIO problem may be impossible to solve in one form and quite easy in another form. However, it is beyond the scope of this manual to discuss good formulations for mixed-integer problems. For discussions on this topic see for example [Wol98].

Chapter 14

Additional features

In this section we describe additional features and tools which enable more detailed analysis of optimization problems with **MOSEK**.

14.1 Problem Analyzer

The problem analyzer prints a survey of the structure of the problem, with information about linear constraints and objective, quadratic constraints, conic constraints and variables.

In the initial stages of model formulation the problem analyzer may be used as a quick way of verifying that the model has been built or imported correctly. In later stages it can help revealing special structures within the model that may be used to tune the optimizer's performance or to identify the causes of numerical difficulties.

The problem analyzer is run using *Task.analyzeproblem*. It prints its output to a log stream. The output is similar to the one below (this is the problem survey of the `aflow30a` problem from the MIPLIB 2003 collection).

```
                   Analyzing the problem

*** Structural report
               Dimensions
               Constraints   Variables     Matrix var.   Cones
               479           842           0             0

               Constraint and bound types
               Free          Lower         Upper         Ranged        Fixed
 Constraints:  0             0             421           0             58
   Variables:  0             0             0             842           0

               Integer constraint types
               Binary        General
               421           0

*** Data report
               Nonzeros      Min           Max
        |cj|:  421           1.1e+01       5.0e+02
       |Aij|:  2091          1.0e+00       1.0e+02

               # finite      Min           Max
      |blci|:  58            1.0e+00       1.0e+01
      |buci|:  479           0.0e+00       1.0e+01
      |blxj|:  842           0.0e+00       0.0e+00
      |buxj|:  842           1.0e+00       1.0e+02
```

(continues on next page)

(continued from previous page)

```
*** Done analyzing the problem
```

The survey is divided into a structural and numerical report. The content should be self-explanatory.

14.2 Automatic Repair of Infeasible Problems

MOSEK provides an automatic repair tool for infeasible linear problems which we cover in this section. Note that most infeasible models are so due to bugs which can (and should) be more reliably fixed manually, using the knowledge of the model structure. We discuss this approach in Sec. 8.3.

14.2.1 Automatic repair

The main idea can be described as follows. Consider the linear optimization problem with m constraints and n variables

$$\begin{array}{lccccc} \text{minimize} & & & c^T x + c^f & & \\ \text{subject to} & l^c & \leq & Ax & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x, \end{array}$$

which is assumed to be infeasible.

One way of making the problem feasible is to reduce the lower bounds and increase the upper bounds. If the change is sufficiently large the problem becomes feasible. Now an obvious idea is to compute the optimal relaxation by solving an optimization problem. The problem

$$\begin{array}{lccccc} \text{minimize} & & & p(v_l^c, v_u^c, v_l^x, v_u^x) & & \\ \text{subject to} & l^c - v_l^c & \leq & Ax & \leq & u^c + v_u^c, \\ & l^x - v_l^x & \leq & x & \leq & u^x + v_u^x, \\ & & & v_l^c, v_u^c, v_l^x, v_u^x \geq 0 & & \end{array} \tag{14.1}$$

does exactly that. The additional variables $(v_l^c)_i$, $(v_u^c)_i$, $(v_l^x)_j$ and $(v_u^c)_j$ are *elasticity* variables because they allow a constraint to be violated and hence add some elasticity to the problem. For instance, the elasticity variable $(v_l^c)_i$ controls how much the lower bound $(l^c)_i$ should be relaxed to make the problem feasible. Finally, the so-called penalty function

$$p(v_l^c, v_u^c, v_l^x, v_u^x)$$

is chosen so it penalizes changes to bounds. Given the weights

- $w_l^c \in \mathbb{R}^m$ (associated with l^c),
- $w_u^c \in \mathbb{R}^m$ (associated with u^c),
- $w_l^x \in \mathbb{R}^n$ (associated with l^x),
- $w_u^x \in \mathbb{R}^n$ (associated with u^x),

a natural choice is

$$p(v_l^c, v_u^c, v_l^x, v_u^x) = (w_l^c)^T v_l^c + (w_u^c)^T v_u^c + (w_l^x)^T v_l^x + (w_u^x)^T v_u^x.$$

Hence, the penalty function $p()$ is a weighted sum of the elasticity variables and therefore the problem (14.1) keeps the amount of relaxation at a minimum. Please observe that

- the problem (14.1) is always feasible.
- a negative weight implies problem (14.1) is unbounded. For this reason if the value of a weight is negative **MOSEK** fixes the associated elasticity variable to zero. Clearly, if one or more of the weights are negative, it may imply that it is not possible to repair the problem.

A simple choice of weights is to set them all to 1, but of course that does not take into account that constraints may have different importance.

Caveats

- Observe if the infeasible problem

$$\begin{array}{lrcl} \text{minimize} & x+z & & \\ \text{subject to} & x & = & -1, \\ & x & \geq & 0 \end{array}$$

  is repaired then it will become unbounded. Hence, a repaired problem may not have an optimal solution.

- Only a minimal repair is performed i.e. the repair that only just makes the problem feasible. Hence, the repaired problem is barely feasible and that sometimes makes the repaired problem hard to solve.

- The infeasibility repair is *unable* to fix crossing bounds, that is ranged constraints of the form $l \leq Ax \leq u$ where $l > u$. In this case it will always fail with an error. Crossing bounds have to be fixed by the user prior to calling automatic repair.

Using the automatic repair tool

In this subsection we consider an infeasible linear optimization example:

$$\begin{array}{lrcrcrl} \text{minimize} & -10x_1 & & -9x_2, & & & \\ \text{subject to} & 7/10x_1 & + & 1x_2 & \leq & 630, & \\ & 1/2x_1 & + & 5/6x_2 & \leq & 600, & \\ & 1x_1 & + & 2/3x_2 & \leq & 708, & \\ & 1/10x_1 & + & 1/4x_2 & \leq & 135, & \\ & x_1, & & x_2 & \geq & 0, & \\ & & & x_2 & \geq & 650. & \end{array} \tag{14.2}$$

The function *Task.primalrepair* can be used to repair an infeasible problem. This can be used for linear and conic optimization problems, possibly with integer variables.

Listing 14.1: An example of feasibility repair applied to problem (14.2).

```
import sys
import mosek

# Since the actual value of Infinity is ignores, we define it solely
# for symbolic purposes:
inf = 0.0

# Define a stream printer to grab output from MOSEK
def streamprinter(text):
    sys.stdout.write(text)
    sys.stdout.flush()


def main(inputfile):
    with mosek.Task() as task:
        # Attach a printer to the task
        task.set_Stream(mosek.streamtype.log, streamprinter)

        # Read data
        task.readdata(inputfile)

        task.putintparam(mosek.iparam.log_feas_repair, 3)
```

(continues on next page)

(continued from previous page)

```
        task.primalrepair(None, None, None, None)

        sum_viol = task.getdouinf(mosek.dinfitem.primal_repair_penalty_obj)
        print("Minimized sum of violations = %e" % sum_viol)

        task.optimize()

        task.solutionsummary(mosek.streamtype.msg)

# call the main function
try:
    filename = "../data/feasrepair.lp"
    if len(sys.argv) > 1:
        filename = sys.argv[1]
    main(filename)
except Exception as e:
    print(e)
    raise
```

The above code will produce the following log report:

```
MOSEK Version 9.0.0.25(ALPHA) (Build date: 2017-11-7 16:11:50)
Copyright (c) MOSEK ApS, Denmark. WWW: mosek.com
Platform: Linux/64-X86

Open file 'feasrepair.lp'
Reading started.
Reading terminated. Time: 0.00

Read summary
  Type             : LO (linear optimization problem)
  Objective sense  : min
  Scalar variables : 2
  Matrix variables : 0
  Constraints      : 4
  Cones            : 0
  Time             : 0.0

Problem
  Name                   :
  Objective sense        : min
  Type                   : LO (linear optimization problem)
  Constraints            : 4
  Cones                  : 0
  Scalar variables       : 2
  Matrix variables       : 0
  Integer variables      : 0

Primal feasibility repair started.
Optimizer started.
Presolve started.
Linear dependency checker started.
Linear dependency checker terminated.
Eliminator started.
Freed constraints in eliminator : 2
Eliminator terminated.
Eliminator - tries                  : 1                 time                   : 0.00
```

(continues on next page)

(continued from previous page)

```
Lin. dep.  - tries                  : 1                 time                   : 0.00
Lin. dep.  - number                 : 0
Presolve terminated. Time: 0.00
Problem
  Name                   :
  Objective sense        : min
  Type                   : LO (linear optimization problem)
  Constraints            : 8
  Cones                  : 0
  Scalar variables       : 14
  Matrix variables       : 0
  Integer variables      : 0

Optimizer  - threads                : 20
Optimizer  - solved problem         : the primal
Optimizer  - Constraints            : 2
Optimizer  - Cones                  : 0
Optimizer  - Scalar variables       : 5                 conic                  : 0
Optimizer  - Semi-definite variables: 0                 scalarized             : 0
Factor     - setup time             : 0.00              dense det. time        : 0.00
Factor     - ML order time          : 0.00              GP order time          : 0.00
Factor     - nonzeros before factor : 3                 after factor           : 3
Factor     - dense dim.             : 0                 flops                  : 5.
↪00e+01
ITE PFEAS    DFEAS    GFEAS    PRSTATUS   POBJ              DOBJ              MU
↪ TIME
0   2.7e+01  1.0e+00  4.0e+00  1.00e+00   3.000000000e+00   0.000000000e+00   1.0e+00␣
↪ 0.00
1   2.5e+01  9.1e-01  1.4e+00  0.00e+00   8.711262850e+00   1.115287830e+01   2.4e+00␣
↪ 0.00
2   2.4e+00  8.8e-02  1.4e-01  -7.33e-01  4.062505701e+01   4.422203730e+01   2.3e-01␣
↪ 0.00
3   9.4e-02  3.4e-03  5.5e-03  1.33e+00   4.250700434e+01   4.258548510e+01   9.1e-03␣
↪ 0.00
4   2.0e-05  7.2e-07  1.1e-06  1.02e+00   4.249996599e+01   4.249998669e+01   1.9e-06␣
↪ 0.00
5   2.0e-09  7.2e-11  1.1e-10  1.00e+00   4.250000000e+01   4.250000000e+01   1.9e-10␣
↪ 0.00
Basis identification started.
Basis identification terminated. Time: 0.00
Optimizer terminated. Time: 0.01

Basic solution summary
  Problem status  : PRIMAL_AND_DUAL_FEASIBLE
  Solution status : OPTIMAL
  Primal.  obj: 4.2500000000e+01    nrm: 6e+02    Viol.  con: 1e-13    var: 0e+00
  Dual.    obj: 4.2499999999e+01    nrm: 2e+00    Viol.  con: 0e+00    var: 9e-11
Optimal objective value of the penalty problem: 4.250000000000e+01

Repairing bounds.
Increasing the upper bound 1.35e+02 on constraint 'c4' (3) with 2.25e+01.
Decreasing the lower bound 6.50e+02 on variable 'x2' (4) with 2.00e+01.
Primal feasibility repair terminated.
Optimizer started.
Optimizer terminated. Time: 0.00
```

(continues on next page)

(continued from previous page)

```
Interior-point solution summary
  Problem status  : PRIMAL_AND_DUAL_FEASIBLE
  Solution status : OPTIMAL
  Primal.  obj: -5.6700000000e+03   nrm: 6e+02    Viol.  con: 0e+00    var: 0e+00
  Dual.    obj: -5.6700000000e+03   nrm: 1e+01    Viol.  con: 0e+00    var: 0e+00

Basic solution summary
  Problem status  : PRIMAL_AND_DUAL_FEASIBLE
  Solution status : OPTIMAL
  Primal.  obj: -5.6700000000e+03   nrm: 6e+02    Viol.  con: 0e+00    var: 0e+00
  Dual.    obj: -5.6700000000e+03   nrm: 1e+01    Viol.  con: 0e+00    var: 0e+00

Optimizer summary
  Optimizer                 -                        time: 0.00
    Interior-point          - iterations : 0         time: 0.00
      Basis identification  -                        time: 0.00
        Primal              - iterations : 0         time: 0.00
        Dual                - iterations : 0         time: 0.00
        Clean primal        - iterations : 0         time: 0.00
        Clean dual          - iterations : 0         time: 0.00
    Simplex                 -                        time: 0.00
      Primal simplex        - iterations : 0         time: 0.00
      Dual simplex          - iterations : 0         time: 0.00
    Mixed integer           - relaxations: 0         time: 0.00
```

It will also modify the task according to the optimal elasticity variables found. In this case the optimal repair it is to increase the upper bound on constraint `c4` by 22.5 and decrease the lower bound on variable `x2` by 20.

14.3 Sensitivity Analysis

Given an optimization problem it is often useful to obtain information about how the optimal objective value changes when the problem parameters are perturbed. E.g, assume that a bound represents the capacity of a machine. Now, it may be possible to expand the capacity for a certain cost and hence it is worthwhile knowing what the value of additional capacity is. This is precisely the type of questions the sensitivity analysis deals with.

Analyzing how the optimal objective value changes when the problem data is changed is called *sensitivity analysis*.

References

The book [Chvatal83] discusses the classical sensitivity analysis in Chapter 10 whereas the book [RTV97] presents a modern introduction to sensitivity analysis. Finally, it is recommended to read the short paper [Wal00] to avoid some of the pitfalls associated with sensitivity analysis.

Warning: Currently, sensitivity analysis is only available for continuous linear optimization problems. Moreover, **MOSEK** can only deal with perturbations of bounds and objective function coefficients.

14.3.1 Sensitivity Analysis for Linear Problems

The Optimal Objective Value Function

Assume that we are given the problem

$$\begin{array}{rcllcll} z(l^c, u^c, l^x, u^x, c) & = & \text{minimize} & & & c^T x & \\ & & \text{subject to} & l^c & \leq & Ax & \leq u^c, \\ & & & l^x & \leq & x & \leq u^x, \end{array} \tag{14.3}$$

and we want to know how the optimal objective value changes as l_i^c is perturbed. To answer this question we define the perturbed problem for l_i^c as follows

$$\begin{array}{rcllll} f_{l_i^c}(\beta) & = & \text{minimize} & & c^T x & \\ & & \text{subject to} & l^c + \beta e_i & \leq Ax & \leq u^c, \\ & & & l^x & \leq x \leq & u^x, \end{array}$$

where e_i is the i-th column of the identity matrix. The function

$$f_{l_i^c}(\beta) \tag{14.4}$$

shows the optimal objective value as a function of β. Please note that a change in β corresponds to a perturbation in l_i^c and hence (14.4) shows the optimal objective value as a function of varying l_i^c with the other bounds fixed.

It is possible to prove that the function (14.4) is a piecewise linear and convex function, i.e. its graph may look like in Fig. 14.1 and Fig. 14.2.

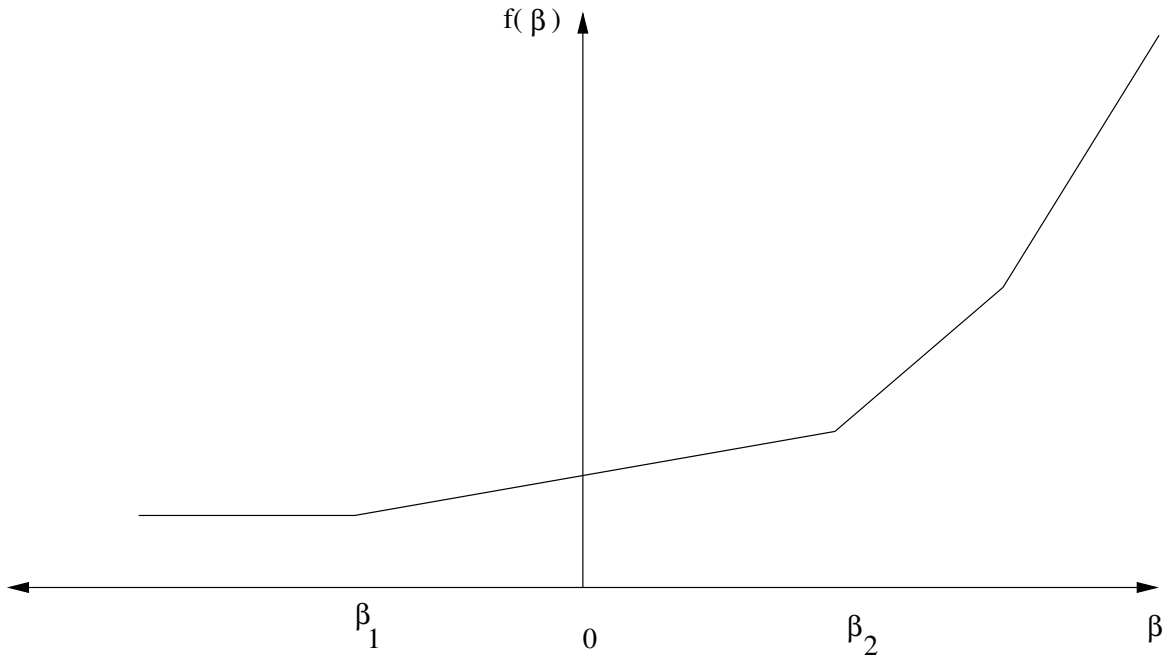

Fig. 14.1: $\beta = 0$ is in the interior of linearity interval.

Clearly, if the function $f_{l_i^c}(\beta)$ does not change much when β is changed, then we can conclude that the optimal objective value is insensitive to changes in l_i^c. Therefore, we are interested in the rate of change in $f_{l_i^c}(\beta)$ for small changes in β — specifically the gradient

$$f'_{l_i^c}(0),$$

which is called the *shadow price* related to l_i^c. The shadow price specifies how the objective value changes for small changes of β around zero. Moreover, we are interested in the *linearity interval*

$$\beta \in [\beta_1, \beta_2]$$

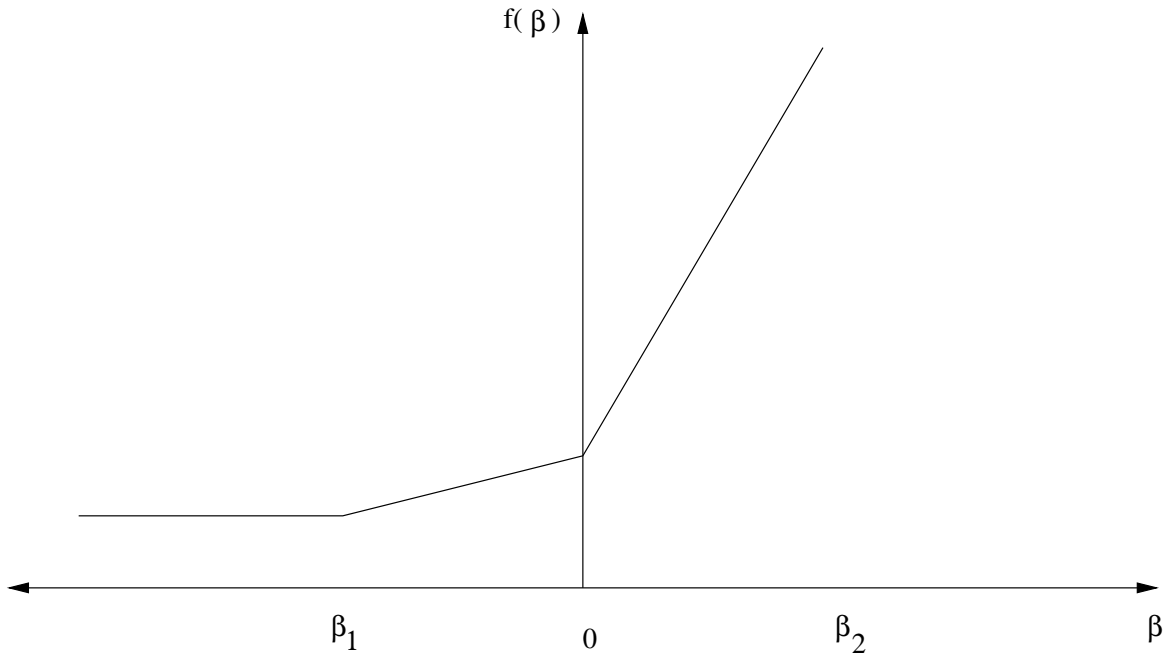

Fig. 14.2: $\beta = 0$ is a breakpoint.

for which

$$f'_{l_i^c}(\beta) = f'_{l_i^c}(0).$$

Since $f_{l_i^c}$ is not a smooth function $f'_{l_i^c}$ may not be defined at 0, as illustrated in Fig. 14.2. In this case we can define a left and a right shadow price and a left and a right linearity interval.

The function $f_{l_i^c}$ considered only changes in l_i^c. We can define similar functions for the remaining parameters of the z defined in (14.3) as well:

$$\begin{array}{rcll} f_{l_i^c}(\beta) & = & z(l^c + \beta e_i, u^c, l^x, u^x, c), & i = 1, \ldots, m, \\ f_{u_i^c}(\beta) & = & z(l^c, u^c + \beta e_i, l^x, u^x, c), & i = 1, \ldots, m, \\ f_{l_j^x}(\beta) & = & z(l^c, u^c, l^x + \beta e_j, u^x, c), & j = 1, \ldots, n, \\ f_{u_j^x}(\beta) & = & z(l^c, u^c, l^x, u^x + \beta e_j, c), & j = 1, \ldots, n, \\ f_{c_j}(\beta) & = & z(l^c, u^c, l^x, u^x, c + \beta e_j), & j = 1, \ldots, n. \end{array}$$

Given these definitions it should be clear how linearity intervals and shadow prices are defined for the parameters u_i^c etc.

Equality Constraints

In **MOSEK** a constraint can be specified as either an equality constraint or a ranged constraint. If some constraint e_i^c is an equality constraint, we define the optimal value function for this constraint as

$$f_{e_i^c}(\beta) = z(l^c + \beta e_i, u^c + \beta e_i, l^x, u^x, c)$$

Thus for an equality constraint the upper and the lower bounds (which are equal) are perturbed simultaneously. Therefore, **MOSEK** will handle sensitivity analysis differently for a ranged constraint with $l_i^c = u_i^c$ and for an equality constraint.

The Basis Type Sensitivity Analysis

The classical sensitivity analysis discussed in most textbooks about linear optimization, e.g. [Chvatal83], is based on an optimal basis. This method may produce misleading results [RTV97] but is computationally cheap. This is the type of sensitivity analysis implemented in **MOSEK**.

We will now briefly discuss the basis type sensitivity analysis. Given an optimal basic solution which provides a partition of variables into basic and non-basic variables, the basis type sensitivity analysis computes the linearity interval $[\beta_1, \beta_2]$ so that the basis remains optimal for the perturbed problem. A shadow price associated with the linearity interval is also computed. However, it is well-known that an optimal basic solution may not be unique and therefore the result depends on the optimal basic solution employed in the sensitivity analysis. If the optimal objective value function has a breakpoint for $\beta = 0$ then the basis type sensitivity method will only provide a subset of either the left or the right linearity interval.

In summary, the basis type sensitivity analysis is computationally cheap but does not provide complete information. Hence, the results of the basis type sensitivity analysis should be used with care.

Example: Sensitivity Analysis

As an example we will use the following transportation problem. Consider the problem of minimizing the transportation cost between a number of production plants and stores. Each plant supplies a number of goods and each store has a given demand that must be met. Supply, demand and cost of transportation per unit are shown in Fig. 14.3.

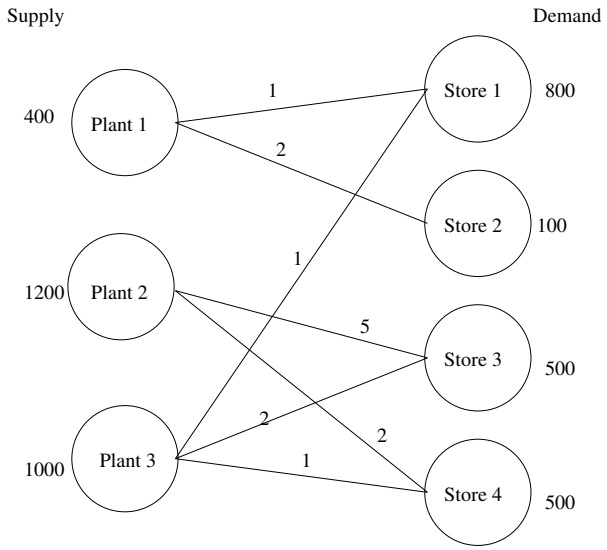

Fig. 14.3: Supply, demand and cost of transportation.

If we denote the number of transported goods from location i to location j by x_{ij}, problem can be formulated as the linear optimization problem of minimizing

$$1x_{11} + 2x_{12} + 5x_{23} + 2x_{24} + 1x_{31} + 2x_{33} + 1x_{34}$$

subject to

$$\begin{array}{rcrcrcrcrcrcl}
x_{11} & + & x_{12} & & & & & & & & & \leq & 400, \\
 & & & x_{23} & + & x_{24} & & & & & & \leq & 1200, \\
 & & & & & & x_{31} & + & x_{33} & + & x_{34} & \leq & 1000, \\
x_{11} & & & & & & + & x_{31} & & & & = & 800, \\
 & & x_{12} & & & & & & & & & = & 100, \\
 & & & x_{23} & + & & & & x_{33} & & & = & 500, \\
 & & & & & x_{24} & + & & & & x_{34} & = & 500, \\
x_{11}, & & x_{12}, & & x_{23}, & & x_{24}, & & x_{31}, & & x_{33}, & & x_{34} \geq 0.
\end{array} \tag{14.5}$$

The sensitivity parameters are shown in Table 14.1 and Table 14.2.

Table 14.1: Ranges and shadow prices related to bounds on constraints and variables.

Con.	β_1	β_2	σ_1	σ_2
1	-300.00	0.00	3.00	3.00
2	-700.00	$+\infty$	0.00	0.00
3	-500.00	0.00	3.00	3.00
4	-0.00	500.00	4.00	4.00
5	-0.00	300.00	5.00	5.00
6	-0.00	700.00	5.00	5.00
7	-500.00	700.00	2.00	2.00
Var.	β_1	β_2	σ_1	σ_2
x_{11}	$-\infty$	300.00	0.00	0.00
x_{12}	$-\infty$	100.00	0.00	0.00
x_{23}	$-\infty$	0.00	0.00	0.00
x_{24}	$-\infty$	500.00	0.00	0.00
x_{31}	$-\infty$	500.00	0.00	0.00
x_{33}	$-\infty$	500.00	0.00	0.00
x_{34}	-0.000000	500.00	2.00	2.00

Table 14.2: Ranges and shadow prices related to the objective coefficients.

Var.	β_1	β_2	σ_1	σ_2
c_1	$-\infty$	3.00	300.00	300.00
c_2	$-\infty$	∞	100.00	100.00
c_3	-2.00	∞	0.00	0.00
c_4	$-\infty$	2.00	500.00	500.00
c_5	-3.00	∞	500.00	500.00
c_6	$-\infty$	2.00	500.00	500.00
c_7	-2.00	∞	0.00	0.00

Examining the results from the sensitivity analysis we see that for constraint number 1 we have $\sigma_1 = 3$ and $\beta_1 = -300,\ \beta_2 = 0$.

If the upper bound on constraint 1 is decreased by

$$\beta \in [0, 300]$$

then the optimal objective value will increase by the value

$$\sigma_1 \beta = 3\beta.$$

14.3.2 Sensitivity Analysis with MOSEK

MOSEK provides the functions *Task.primalsensitivity* and *Task.dualsensitivity* for performing sensitivity analysis. The code in Listing 14.2 gives an example of its use.

Listing 14.2: Example of sensitivity analysis with the **MOSEK** Optimizer API for Python.

```
import sys
import mosek

# Since the actual value of Infinity is ignores, we define it solely
# for symbolic purposes:
inf = 0.0

# Define a stream printer to grab output from MOSEK
def streamprinter(text):
    sys.stdout.write(text)
    sys.stdout.flush()


def main():
    # Create a task
    with mosek.Task() as task:
        # Attach a printer to the task
        task.set_Stream(mosek.streamtype.log, streamprinter)

        # Set up data
        bkc = [mosek.boundkey.up, mosek.boundkey.up,
               mosek.boundkey.up, mosek.boundkey.fx,
               mosek.boundkey.fx, mosek.boundkey.fx,
               mosek.boundkey.fx]
        blc = [-inf, -inf, -inf, 800., 100., 500., 500.]
        buc = [400., 1200., 1000., 800., 100., 500., 500.]

        bkx = [mosek.boundkey.lo, mosek.boundkey.lo,
               mosek.boundkey.lo, mosek.boundkey.lo,
               mosek.boundkey.lo, mosek.boundkey.lo,
               mosek.boundkey.lo]
        c = [1.0, 2.0, 5.0, 2.0, 1.0, 2.0, 1.0]
        blx = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
        bux = [inf, inf, inf, inf, inf, inf, inf]

        ptrb = [0, 2, 4, 6, 8, 10, 12]
        ptre = [2, 4, 6, 8, 10, 12, 14]
        sub = [0, 3, 0, 4, 1, 5, 1, 6, 2, 3, 2, 5, 2, 6]

        val = [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
               1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]

        numcon = len(bkc)
        numvar = len(bkx)
        numanz = len(val)

        # Input linear data
        task.inputdata(numcon, numvar,
                       c, 0.0,
                       ptrb, ptre, sub, val,
```

(continues on next page)

(continued from previous page)

```
                        bkc, blc, buc,
                        bkx, blx, bux)
        # Set objective sense
        task.putobjsense(mosek.objsense.minimize)

        # Optimize
        task.optimize()

        # Analyze upper bound on c1 and the equality constraint on c4
        subi = [0, 3]
        marki = [mosek.mark.up, mosek.mark.up]

        # Analyze lower bound on the variables x12 and x31
        subj = [1, 4]
        markj = [mosek.mark.lo, mosek.mark.lo]

        (leftpricei, rightpricei, leftrangei, rightrangei,
         leftpricej, rightpricej, leftrangej, rightrangej) = task.
↪primalsensitivity(subi,
                                                                    ␣
↪marki,
                                                                    ␣
↪subj,
                                                                    ␣
↪markj)

        print('Results from sensitivity analysis on bounds:')
        print('\tleftprice  | rightprice | leftrange  | rightrange ')
        print('For constraints:')

        for i in range(2):
            print('\t%10f   %10f   %10f   %10f' % (leftpricei[i],
                                                   rightpricei[i],
                                                   leftrangei[i],
                                                   rightrangei[i]))

        print('For variables:')
        for i in range(2):
            print('\t%10f   %10f   %10f   %10f' % (leftpricej[i],
                                                   rightpricej[i],
                                                   leftrangej[i],
                                                   rightrangej[i]))

        subc = [2, 5]

        leftprice, rightprice, leftrange, rightrange = task.dualsensitivity(subc)

        print('Results from sensitivity analysis on objective coefficients:')

        for i in range(2):
            print('\t%10f   %10f   %10f   %10f' % (leftprice[i],
                                                   rightprice[i],
                                                   leftrange[i],
                                                   rightrange[i]))

    return None
```

(continues on next page)

(continued from previous page)

```
# call the main function
try:
    main()
except mosek.MosekException as e:
    print("ERROR: %s" % str(e.errno))
    if e.msg is not None:
        print("\t%s" % e.msg)
    sys.exit(1)
except:
    import traceback
    traceback.print_exc()
    sys.exit(1)
```

Chapter 15

API Reference

This section contains the complete reference of the **MOSEK** Optimizer API for Python. It is organized as follows:

- *General API conventions.*
- **Methods:**
  - *Class Env* (The **MOSEK** environment)
  - *Class Task* (An optimization task)
  - *Browse by topic*
- **Optimizer parameters:**
  - *Double*, *Integer*, *String*
  - *Full list*
  - *Browse by topic*
- **Optimizer information items:**
  - `Double`, `Integer`, `Long`
- *Optimizer response codes*
- *Enumerations*
- *Exceptions*
- *User-defined function types*
- *List of supported domains*
- *Environment variables*

15.1 API Conventions

15.1.1 Function arguments

Naming Convention

In the definition of the **MOSEK** Optimizer API for Python a consistent naming convention has been used. This implies that whenever for example `numcon` is an argument in a function definition it indicates the number of constraints. In Table 15.1 the variable names used to specify the problem parameters are listed.

Table 15.1: Naming conventions used in the **MOSEK** Optimizer API for Python.

API name	API type	Dimension	Related problem parameter
numcon	int		m
numvar	int		n
numcone	int		t
aptrb	int[]	numvar	a_{ij}
aptre	int[]	numvar	a_{ij}
asub	int[]	aptre[numvar-1]	a_{ij}
aval	float[]	aptre[numvar-1]	a_{ij}
c	float[]	numvar	c_j
cfix	float		c^f
blc	float[]	numcon	l_k^c
buc	float[]	numcon	u_k^c
blx	float[]	numvar	l_k^x
bux	float[]	numvar	u_k^x
numqonz	int		q_{ij}^o
qosubi	int[]	numqonz	q_{ij}^o
qosubj	int[]	numqonz	q_{ij}^o
qoval	float[]	numqonz	q_{ij}^o
numqcnz	int		q_{ij}^k
qcsubk	int[]	numqcnz	q_{ij}^k
qcsubi	int[]	numqcnz	q_{ij}^k
qcsubj	int[]	numqcnz	q_{ij}^k
qcval	float[]	numqcnz	q_{ij}^k
bkc	int[]	numcon	l_k^c and u_k^c
bkx	int[]	numvar	l_k^x and u_k^x

The relation between the variable names and the problem parameters is as follows:

- The quadratic terms in the objective: $q^o_{\texttt{qosubi[t]},\texttt{qosubj[t]}} = \texttt{qoval[t]}, \quad t = 0, \ldots, \texttt{numqonz} - 1$.
- The linear terms in the objective : $c_j = \texttt{c[j]}, \quad j = 0, \ldots, \texttt{numvar} - 1$
- The fixed term in the objective : $c^f = \texttt{cfix}$.
- The quadratic terms in the constraints: $q^{\texttt{qcsubk[t]}}_{\texttt{qcsubi[t]},\texttt{qcsubj[t]}} = \texttt{qcval[t]}, \quad t = 0, \ldots, \texttt{numqcnz} - 1$
- The linear terms in the constraints: $a_{\texttt{asub[t]},\texttt{j}} = \texttt{aval[t]}, \quad t = \texttt{ptrb[j]}, \ldots, \texttt{ptre[j]} - 1, \; j = 0, \ldots, \texttt{numvar} - 1$

Information about input/output arguments

The following are purely informational tags which indicate how **MOSEK** treats a specific function argument.

- (input) An input argument. It is used to input data to **MOSEK**.
- (output) An output argument. It can be a user-preallocated data structure, a reference, a string buffer etc. where **MOSEK** will output some data.
- (input/output) An input/output argument. **MOSEK** will read the data and overwrite it with new/updated information.

15.1.2 Bounds

The bounds on the constraints and variables are specified using the variables `bkc`, `blc`, and `buc`. The components of the integer array `bkc` specify the bound type according to Table 15.2

Table 15.2: Symbolic key for variable and constraint bounds.

Symbolic constant	Lower bound	Upper bound
boundkey.fx	finite	identical to the lower bound
boundkey.fr	minus infinity	plus infinity
boundkey.lo	finite	plus infinity
boundkey.ra	finite	finite
boundkey.up	minus infinity	finite

For instance `bkc[2]=`*boundkey.lo* means that $-\infty < l_2^c$ and $u_2^c = \infty$. Even if a variable or constraint is bounded only from below, e.g. $x \geq 0$, both bounds are inputted or extracted; the irrelevant value is ignored.

Finally, the numerical values of the bounds are given by

$$l_k^c = \texttt{blc}[\texttt{k}], \quad k = 0, \ldots, \texttt{numcon} - 1$$

$$u_k^c = \texttt{buc}[\texttt{k}], \quad k = 0, \ldots, \texttt{numcon} - 1.$$

The bounds on the variables are specified using the variables `bkx`, `blx`, and `bux` in the same way. The numerical values for the lower bounds on the variables are given by

$$l_j^x = \texttt{blx}[\texttt{j}], \quad j = 0, \ldots, \texttt{numvar} - 1.$$

$$u_j^x = \texttt{bux}[\texttt{j}], \quad j = 0, \ldots, \texttt{numvar} - 1.$$

15.1.3 Vector Formats

Three different vector formats are used in the **MOSEK** API:

Full (dense) vector

This is simply an array where the first element corresponds to the first item, the second element to the second item etc. For example to get the linear coefficients of the objective in `task` with `numvar` variables, one would write

```
c = zeros(numvar,float)
task.getc(c)
```

Vector slice

A vector slice is a range of values from `first` up to and **not including** `last` entry in the vector, i.e. for the set of indices `i` such that `first <= i < last`. For example, to get the bounds associated with constrains 2 through 9 (both inclusive) one would write

```
upper_bound = zeros(8,float)
lower_bound = zeros(8,float)
bound_key   = array([0] * 8)

task.getconboundslice(2, 10,
                      bound_key,lower_bound,upper_bound)
```

Sparse vector

A sparse vector is given as an array of indexes and an array of values. The indexes need not be ordered. For example, to input a set of bounds associated with constraints number 1, 6, 3, and 9, one might write

```
bound_index = [          1,          6,          3,          9]
bound_key   = [boundkey.fr,boundkey.lo,boundkey.up,boundkey.fx]
lower_bound = [     0.0,         -10.0,        0.0,        5.0]
upper_bound = [     0.0,           0.0,        6.0,        5.0]
task.putconboundlist(bound_index,
                     bound_key,lower_bound,upper_bound)
```

15.1.4 Matrix Formats

The coefficient matrices in a problem are inputted and extracted in a sparse format. That means only the nonzero entries are listed.

Unordered Triplets

In unordered triplet format each entry is defined as a row index, a column index and a coefficient. For example, to input the A matrix coefficients for $a_{1,2} = 1.1, a_{3,3} = 4.3$, and $a_{5,4} = 0.2$, one would write as follows:

```
subi = [   1,   3,   5 ]
subj = [   2,   3,   4 ]
cof  = [ 1.1, 4.3, 0.2 ]
task.putaijlist(subi,subj,cof)
```

Please note that in some cases (like *Task.putaijlist*) *only* the specified indexes are modified — all other are unchanged. In other cases (such as *Task.putqconk*) the triplet format is used to modify *all* entries — entries that are not specified are set to 0.

Column or Row Ordered Sparse Matrix

In a sparse matrix format only the non-zero entries of the matrix are stored. **MOSEK** uses a sparse packed matrix format ordered either by columns or rows. Here we describe the column-wise format. The row-wise format is based on the same principle.

Column ordered sparse format

A sparse matrix in column ordered format is essentially a list of all non-zero entries read column by column from left to right and from top to bottom within each column. The exact representation uses four arrays:

- `asub`: Array of size equal to the number of nonzeros. List of row indexes.
- `aval`: Array of size equal to the number of nonzeros. List of non-zero entries of A ordered by columns.
- `ptrb`: Array of size `numcol`, where `ptrb[j]` is the position of the first value/index in `aval`/ `asub` for the j-th column.
- `ptre`: Array of size `numcol`, where `ptre[j]` is the position of the last value/index plus one in `aval` / `asub` for the j-th column.

With this representation the values of a matrix A with `numcol` columns are assigned using:

$$a_{\texttt{asub}[k],j} = \texttt{aval}[k] \quad \text{for} \quad j = 0, \ldots, \texttt{numcol} - 1 - 1, \;\; k = \texttt{ptrb}[j], \ldots, \texttt{ptre}[j] - 1.$$

As an example consider the matrix

$$A = \begin{bmatrix} 1.1 & & 1.3 & 1.4 & \\ & 2.2 & & & 2.5 \\ 3.1 & & & 3.4 & \\ & & 4.4 & & \end{bmatrix} \tag{15.1}$$

which can be represented in the column ordered sparse matrix format as

$$\begin{array}{lcl} \texttt{ptrb} & = & [0,2,3,5,7], \\ \texttt{ptre} & = & [2,3,5,7,8], \\ \texttt{asub} & = & [0,2,1,0,3,0,2,1], \\ \texttt{aval} & = & [1.1,3.1,2.2,1.3,4.4,1.4,3.4,2.5]. \end{array}$$

Fig. 15.1 illustrates how the matrix A in (15.1) is represented in column ordered sparse matrix format.

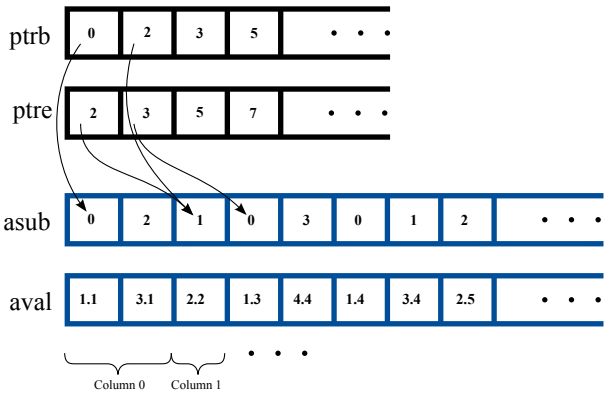

Fig. 15.1: The matrix A (15.1) represented in column ordered packed sparse matrix format.

Column ordered sparse format with nonzeros

Note that `nzc[j] := ptre[j]-ptrb[j]` is exactly the number of nonzero elements in the j-th column of A. In some functions a sparse matrix will be represented using the equivalent dataset `asub`, `aval`, `ptrb`, `nzc`. The matrix A (15.1) would now be represented as:

$$\begin{array}{lcl} \texttt{ptrb} & = & [0,2,3,5,7], \\ \texttt{nzc} & = & [2,1,2,2,1], \\ \texttt{asub} & = & [0,2,1,0,3,0,2,1], \\ \texttt{aval} & = & [1.1,3.1,2.2,1.3,4.4,1.4,3.4,2.5]. \end{array}$$

Row ordered sparse matrix

The matrix A (15.1) can also be represented in the row ordered sparse matrix format as:

$$\begin{array}{lcl} \texttt{ptrb} & = & [0,3,5,7], \\ \texttt{ptre} & = & [3,5,7,8], \\ \texttt{asub} & = & [0,2,3,1,4,0,3,2], \\ \texttt{aval} & = & [1.1,1.3,1.4,2.2,2.5,3.1,3.4,4.4]. \end{array}$$

15.2 Functions grouped by topic

Callback

- *Task.set_InfoCallback* – Receive callbacks with solver status and information during optimization.
- *Task.set_Progress* – Receive callbacks about current status of the solver during optimization.
- *Task.set_Stream* – Directs all output from a task stream to a callback function.
- *Infrequent:* *Env.set_Stream*

Environment and task management

- *Env.Env* – Constructor of a new environment.
- *Task.Task* – Constructor of a new optimization task.
- *Env.Task* – Creates a new task.
- *Task.getdualproblem* – Obtains the dual problem.
- *Task.puttaskname* – Assigns a new name to the task.
- *Infrequent:* *Task.__del__*, *Env.__del__*, *Task.commitchanges*, *Task.deletesolution*, *Task.putmaxnumacc*, *Task.putmaxnumafe*, *Task.putmaxnumanz*, *Task.putmaxnumbarvar*, *Task.putmaxnumcon*, *Task.putmaxnumdjc*, *Task.putmaxnumdomain*, *Task.putmaxnumqnz*, *Task.putmaxnumvar*, *Task.resizetask*
- *Deprecated:* ~~*Task.putmaxnumcone*~~

Infeasibility diagnostic

- *Task.getinfeasiblesubproblem* – Obtains an infeasible subproblem.
- *Task.infeasibilityreport* – Prints the infeasibility report to an output stream.
- *Task.primalrepair* – Repairs a primal infeasible optimization problem by adjusting the bounds on the constraints and variables.

Information items and statistics

- *Task.getdouinf* – Obtains a double information item.
- *Task.getintinf* – Obtains an integer information item.
- *Task.getlintinf* – Obtains a long integer information item.
- *Task.updatesolutioninfo* – Update the information items related to the solution.

Input/Output

- *Task.writedata* – Writes problem data to a file.
- *Task.writesolution* – Write a solution to a file.
- *Infrequent:* *Task.readbsolution*, *Task.readdata*, *Task.readdataformat*, *Task.readjsonsol*, *Task.readjsonstring*, *Task.readlpstring*, *Task.readopfstring*, *Task.readparamfile*, *Task.readptfstring*, *Task.readsolution*, *Task.readsolutionfile*, *Task.readsummary*, *Task.readtask*, *Task.writebsolution*, *Task.writedatastream*, *Task.writejsonsol*, *Task.writeparamfile*, *Task.writesolutionfile*, *Task.writetask*

Inspecting the task

- *Task.analyzeproblem* – Analyze the data of a task.
- *Task.getnumcon* – Obtains the number of constraints.
- *Task.getnumvar* – Obtains the number of variables.

- *Infrequent:* *Task.analyzesolution*, *Task.getaccafeidxlist*, *Task.getaccb*, *Task.getaccbarfnumblocktriplets*, *Task.getaccdomain*, *Task.getaccfnumnz*, *Task.getaccftrip*, *Task.getaccgvector*, *Task.getaccn*, *Task.getaccname*, *Task.getaccnamelen*, *Task.getaccntot*, *Task.getaccs*, *Task.getacol*, *Task.getacolnumnz*, *Task.getacolslice*, *Task.getacolslicenumnz*, *Task.getacolslicetrip*, *Task.getafebarfnumblocktriplets*, *Task.getafebarfnumrowentries*, *Task.getafebarfrow*, *Task.getafebarfrowinfo*, *Task.getafefnumnz*, *Task.getafefrow*, *Task.getafefrownumnz*, *Task.getafeftrip*, *Task.getafeg*, *Task.getafegslice*, *Task.getaij*, *Task.getapiecenumnz*, *Task.getarow*, *Task.getarownumnz*, *Task.getarowslice*, *Task.getarowslicenumnz*, *Task.getarowslicetrip*, *Task.getatrip*, *Task.getbarablocktriplet*, *Task.getbaraidx*, *Task.getbaraidxij*, *Task.getbaraidxinfo*, *Task.getbarasparsity*, *Task.getbarcblocktriplet*, *Task.getbarcidx*, *Task.getbarcidxinfo*, *Task.getbarcidxj*, *Task.getbarcsparsity*, *Task.getbarvarname*, *Task.getbarvarnameindex*, *Task.getbarvarnamelen*, *Task.getc*, *Task.getcfix*, *Task.getcj*, *Task.getclist*, *Task.getconbound*, *Task.getconboundslice*, *Task.getconname*, *Task.getconnameindex*, *Task.getconnamelen*, *Task.getcslice*, *Task.getdimbarvarj*, *Task.getdjcafeidxlist*, *Task.getdjcb*, *Task.getdjcdomainidxlist*, *Task.getdjcname*, *Task.getdjcnamelen*, *Task.getdjcnumafe*, *Task.getdjcnumafetot*, *Task.getdjcnumdomain*, *Task.getdjcnumdomaintot*, *Task.getdjcnumterm*, *Task.getdjcnumtermtot*, *Task.getdjcs*, *Task.getdjctermsizelist*, *Task.getdomainn*, *Task.getdomainname*, *Task.getdomainnamelen*, *Task.getdomaintype*, *Task.getlenbarvarj*, *Task.getmaxnumanz*, *Task.getmaxnumbarvar*, *Task.getmaxnumcon*, *Task.getmaxnumqnz*, *Task.getmaxnumvar*, *Task.getnumacc*, *Task.getnumafe*, *Task.getnumanz*, *Task.getnumbarablocktriplets*, *Task.getnumbaranz*, *Task.getnumbarcblocktriplets*, *Task.getnumbarcnz*, *Task.getnumbarvar*, *Task.getnumdjc*, *Task.getnumdomain*, *Task.getnumintvar*, *Task.getnumparam*, *Task.getnumqconknz*, *Task.getnumqobjnz*, *Task.getnumsymmat*, *Task.getobjname*, *Task.getobjnamelen*, *Task.getpowerdomainalpha*, *Task.getpowerdomaininfo*, *Task.getprobtype*, *Task.getqconk*, *Task.getqobj*, *Task.getqobjij*, *Task.getsparsesymmat*, *Task.getsymmatinfo*, *Task.gettaskname*, *Task.gettasknamelen*, *Task.getvarbound*, *Task.getvarboundslice*, *Task.getvarname*, *Task.getvarnameindex*, *Task.getvarnamelen*, *Task.getvartype*, *Task.getvartypelist*, *Task.readsummary*
- *Deprecated:* ~~*Task.getcone*~~, ~~*Task.getconeinfo*~~, ~~*Task.getconename*~~, ~~*Task.getconenameindex*~~, ~~*Task.getconenamelen*~~, ~~*Task.getmaxnumcone*~~, ~~*Task.getnumcone*~~, ~~*Task.getnumconemem*~~

License system

- *Env.checkoutlicense* – Check out a license feature from the license server ahead of time.
- *Env.putlicensedebug* – Enables debug information for the license system.
- *Env.putlicensepath* – Set the path to the license file.
- *Env.putlicensewait* – Control whether mosek should wait for an available license if no license is available.
- *Infrequent:* *Env.checkinall*, *Env.checkinlicense*, *Env.expirylicenses*, *Env.licensecleanup*, *Env.putlicensecode*, *Env.resetexpirylicenses*

Linear algebra

- *Infrequent:* *Env.axpy*, *Env.computesparsecholesky*, *Env.dot*, *Env.gemm*, *Env.gemv*, *Env.potrf*, *Env.sparsetriangularsolvedense*, *Env.syeig*, *Env.syevd*, *Env.syrk*

Logging

- *Task.linkfiletostream* – Directs all output from a task stream to a file.
- *Task.onesolutionsummary* – Prints a short summary of a specified solution.
- *Task.optimizersummary* – Prints a short summary with optimizer statistics from last optimization.
- *Task.set_Stream* – Directs all output from a task stream to a callback function.
- *Task.solutionsummary* – Prints a short summary of the current solutions.
- *Infrequent:* *Env.echointro*, *Env.linkfiletostream*, *Env.set_Stream*

Names

- *Env.getcodedesc* – Obtains a short description of a response code.
- *Task.putaccname* – Sets the name of an affine conic constraint.
- *Task.putbarvarname* – Sets the name of a semidefinite variable.
- *Task.putconname* – Sets the name of a constraint.
- *Task.putdjcname* – Sets the name of a disjunctive constraint.
- *Task.putdomainname* – Sets the name of a domain.
- *Task.putobjname* – Assigns a new name to the objective.
- *Task.puttaskname* – Assigns a new name to the task.
- *Task.putvarname* – Sets the name of a variable.
- *Infrequent:* *Task.analyzenames*, *Task.generateaccnames*, *Task.generatebarvarnames*, *Task.generateconnames*, *Task.generatedjcnames*, *Task.generatevarnames*, *Task.getaccname*, *Task.getaccnamelen*, *Task.getbarvarname*, *Task.getbarvarnameindex*, *Task.getbarvarnamelen*, *Task.getconname*, *Task.getconnameindex*, *Task.getconnamelen*, *Task.getdjcname*, *Task.getdjcnamelen*, *Task.getdomainname*, *Task.getdomainnamelen*, *Task.getobjname*, *Task.getobjnamelen*, *Task.getstrparam*, *Task.getstrparamlen*, *Task.gettaskname*, *Task.gettasknamelen*, *Task.getvarname*, *Task.getvarnameindex*, *Task.getvarnamelen*, *Task.isdouparname*, *Task.isintparname*, *Task.isstrparname*, *Task.strtosk*
- *Deprecated:* ~~*Task.generateconenames*~~, ~~*Task.getconename*~~, ~~*Task.getconenameindex*~~, ~~*Task.getconenamelen*~~, ~~*Task.putconename*~~, ~~*Task.strtoconetype*~~

Optimization

- *Task.optimize* – Optimizes the problem.
- *Env.optimizebatch* – Optimize a number of tasks in parallel using a specified number of threads.

Parameters

- *Task.putdouparam* – Sets a double parameter.
- *Task.putintparam* – Sets an integer parameter.
- *Task.putlintparam* – Sets an integer parameter.
- *Task.putparam* – Modifies the value of parameter.
- *Task.putstrparam* – Sets a string parameter.
- *Task.resetdouparam* – Resets a double parameter to its default value.
- *Task.resetintparam* – Resets an integer parameter to its default value.
- *Task.resetparameters* – Resets all parameter values.
- *Task.resetstrparam* – Resets a string parameter to its defalt value.
- *Infrequent:* *Task.getatruncatetol*, *Task.getdouparam*, *Task.getintparam*, *Task.getlintparam*, *Task.getnumparam*, *Task.getstrparam*, *Task.getstrparamlen*, *Task.isdouparname*, *Task.isintparname*, *Task.isstrparname*, *Task.putnadouparam*, *Task.putnaintparam*, *Task.putnastrparam*, *Task.readparamfile*, *Task.writeparamfile*

Problem data - affine conic constraints

- *Task.appendacc* – Appends an affine conic constraint to the task.
- *Task.getaccdoty* – Obtains the doty vector for an affine conic constraint.
- *Task.putaccname* – Sets the name of an affine conic constraint.
- *Infrequent:* *Task.appendaccs*, *Task.appendaccseq*, *Task.appendaccsseq*, *Task.evaluateacc*, *Task.evaluateaccs*, *Task.getaccafeidxlist*, *Task.getaccb*, *Task.getaccbarfnumblocktriplets*, *Task.getaccdomain*, *Task.getaccdotys*, *Task.getaccfnumnz*, *Task.getaccftrip*, *Task.getaccgvector*, *Task.getaccn*, *Task.getaccname*, *Task.getaccnamelen*, *Task.getaccntot*, *Task.getaccs*, *Task.getnumacc*, *Task.putacc*, *Task.putaccb*, *Task.putaccbj*, *Task.putaccdoty*, *Task.putacclist*, *Task.putmaxnumacc*

Problem data - affine expressions

- *Task.appendafes* – Appends a number of empty affine expressions to the optimization task.
- *Task.putafebarfentry* – Inputs one entry in barF.
- *Task.putafebarfentrylist* – Inputs a list of entries in barF.
- *Task.putafebarfrow* – Inputs a row of barF.
- *Task.putafefcol* – Replaces all elements in one column of the F matrix in the affine expressions.
- *Task.putafefentry* – Replaces one entry in F.
- *Task.putafefentrylist* – Replaces a list of entries in F.
- *Task.putafefrow* – Replaces all elements in one row of the F matrix in the affine expressions.
- *Task.putafefrowlist* – Replaces all elements in a number of rows of the F matrix in the affine expressions.
- *Task.putafeg* – Replaces one element in the g vector in the affine expressions.
- *Task.putafegslice* – Modifies a slice of the vector g.

- *Infrequent:* *Task.emptyafebarfrow*, *Task.emptyafebarfrowlist*, *Task.emptyafefcol*, *Task.emptyafefcollist*, *Task.emptyafefrow*, *Task.emptyafefrowlist*, *Task.getaccbarfblocktriplet*, *Task.getafebarfblocktriplet*, *Task.getafebarfnumrowentries*, *Task.getafebarfrow*, *Task.getafebarfrowinfo*, *Task.getafefnumnz*, *Task.getafefrow*, *Task.getafefrownumnz*, *Task.getafeftrip*, *Task.getafeg*, *Task.getafegslice*, *Task.getnumafe*, *Task.putafebarfblocktriplet*, *Task.putafeglist*, *Task.putmaxnumafe*

Problem data - bounds

- *Task.putconbound* – Changes the bound for one constraint.
- *Task.putconboundslice* – Changes the bounds for a slice of the constraints.
- *Task.putvarbound* – Changes the bounds for one variable.
- *Task.putvarboundslice* – Changes the bounds for a slice of the variables.
- *Infrequent:* *Task.chgconbound*, *Task.chgvarbound*, *Task.getconbound*, *Task.getconboundslice*, *Task.getvarbound*, *Task.getvarboundslice*, *Task.inputdata*, *Task.putconboundlist*, *Task.putconboundlistconst*, *Task.putconboundsliceconst*, *Task.putvarboundlist*, *Task.putvarboundlistconst*, *Task.putvarboundsliceconst*

Problem data - cones (deprecated)

- *Deprecated:* ~~*Task.appendcone*~~, ~~*Task.appendconeseq*~~, ~~*Task.appendconesseq*~~, ~~*Task.generateconenames*~~, ~~*Task.getcone*~~, ~~*Task.getconeinfo*~~, ~~*Task.getconename*~~, ~~*Task.getconenameindex*~~, ~~*Task.getconenamelen*~~, ~~*Task.getmaxnumcone*~~, ~~*Task.getnumcone*~~, ~~*Task.getnumconemem*~~, ~~*Task.putcone*~~, ~~*Task.putconename*~~, ~~*Task.putmaxnumcone*~~, ~~*Task.removecones*~~

Problem data - constraints

- *Task.appendcons* – Appends a number of constraints to the optimization task.
- *Task.getnumcon* – Obtains the number of constraints.
- *Task.putconbound* – Changes the bound for one constraint.
- *Task.putconboundslice* – Changes the bounds for a slice of the constraints.
- *Task.putconname* – Sets the name of a constraint.
- *Task.removecons* – Removes a number of constraints.
- *Infrequent:* *Task.chgconbound*, *Task.generateconnames*, *Task.getconbound*, *Task.getconboundslice*, *Task.getconname*, *Task.getconnameindex*, *Task.getconnamelen*, *Task.getmaxnumcon*, *Task.getnumqconknz*, *Task.getqconk*, *Task.inputdata*, *Task.putconboundlist*, *Task.putconboundlistconst*, *Task.putconboundsliceconst*, *Task.putmaxnumcon*

Problem data - disjunctive constraints

- *Task.appenddjcs* – Appends a number of empty disjunctive constraints to the task.
- *Task.putdjc* – Inputs a disjunctive constraint.
- *Task.putdjcname* – Sets the name of a disjunctive constraint.
- *Task.putdjcslice* – Inputs a slice of disjunctive constraints.

- *Infrequent:* *Task.getdjcafeidxlist*, *Task.getdjcb*, *Task.getdjcdomainidxlist*, *Task.getdjcname*, *Task.getdjcnamelen*, *Task.getdjcnumafe*, *Task.getdjcnumafetot*, *Task.getdjcnumdomain*, *Task.getdjcnumdomaintot*, *Task.getdjcnumterm*, *Task.getdjcnumtermtot*, *Task.getdjcs*, *Task.getdjctermsizelist*, *Task.getnumdjc*, *Task.putmaxnumdjc*

Problem data - domain

- *Task.appenddualexpconedomain* – Appends the dual exponential cone domain.
- *Task.appenddualgeomeanconedomain* – Appends the dual geometric mean cone domain.
- *Task.appenddualpowerconedomain* – Appends the dual power cone domain.
- *Task.appenddualpowerconedomainseq* – Appends a sequence of dual power cone domains.
- *Task.appendprimalexpconedomain* – Appends the primal exponential cone domain.
- *Task.appendprimalgeomeanconedomain* – Appends the primal geometric mean cone domain.
- *Task.appendprimalpowerconedomain* – Appends the primal power cone domain.
- *Task.appendprimalpowerconedomainseq* – Appends a sequence of primal power cone domains.
- *Task.appendquadraticconedomain* – Appends the n dimensional quadratic cone domain.
- *Task.appendrdomain* – Appends the n dimensional real number domain.
- *Task.appendrminusdomain* – Appends the n dimensional negative orthant to the list of domains.
- *Task.appendrplusdomain* – Appends the n dimensional positive orthant to the list of domains.
- *Task.appendrquadraticconedomain* – Appends the n dimensional rotated quadratic cone domain.
- *Task.appendrzerodomain* – Appends the n dimensional 0 domain.
- *Task.appendsvecpsdconedomain* – Appends the vectorized SVEC PSD cone domain.
- *Task.putdomainname* – Sets the name of a domain.
- *Infrequent:* *Task.getdomainn*, *Task.getdomainname*, *Task.getdomainnamelen*, *Task.getdomaintype*, *Task.getnumdomain*, *Task.getpowerdomainalpha*, *Task.getpowerdomaininfo*, *Task.putmaxnumdomain*

Problem data - linear part

- *Task.appendcons* – Appends a number of constraints to the optimization task.
- *Task.appendvars* – Appends a number of variables to the optimization task.
- *Task.getnumcon* – Obtains the number of constraints.
- *Task.putacol* – Replaces all elements in one column of the linear constraint matrix.
- *Task.putacolslice* – Replaces all elements in a sequence of columns the linear constraint matrix.
- *Task.putaij* – Changes a single value in the linear coefficient matrix.
- *Task.putaijlist* – Changes one or more coefficients in the linear constraint matrix.
- *Task.putarow* – Replaces all elements in one row of the linear constraint matrix.
- *Task.putarowslice* – Replaces all elements in several rows the linear constraint matrix.
- *Task.putcfix* – Replaces the fixed term in the objective.
- *Task.putcj* – Modifies one linear coefficient in the objective.

- *Task.putconbound* – Changes the bound for one constraint.
- *Task.putconboundslice* – Changes the bounds for a slice of the constraints.
- *Task.putconname* – Sets the name of a constraint.
- *Task.putcslice* – Modifies a slice of the linear objective coefficients.
- *Task.putobjname* – Assigns a new name to the objective.
- *Task.putobjsense* – Sets the objective sense.
- *Task.putvarbound* – Changes the bounds for one variable.
- *Task.putvarboundslice* – Changes the bounds for a slice of the variables.
- *Task.putvarname* – Sets the name of a variable.
- *Task.removecons* – Removes a number of constraints.
- *Task.removevars* – Removes a number of variables.
- *Infrequent:* *Task.chgconbound*, *Task.chgvarbound*, *Task.generatebarvarnames*, *Task.generateconnames*, *Task.generatevarnames*, *Task.getacol*, *Task.getacolnumnz*, *Task.getacolslice*, *Task.getacolslicenumnz*, *Task.getacolslicetrip*, *Task.getaij*, *Task.getapiecenumnz*, *Task.getarow*, *Task.getarownumnz*, *Task.getarowslice*, *Task.getarowslicenumnz*, *Task.getarowslicetrip*, *Task.getatrip*, *Task.getatruncatetol*, *Task.getc*, *Task.getcfix*, *Task.getcj*, *Task.getclist*, *Task.getconbound*, *Task.getconboundslice*, *Task.getconname*, *Task.getconnameindex*, *Task.getconnamelen*, *Task.getcslice*, *Task.getmaxnumanz*, *Task.getmaxnumcon*, *Task.getmaxnumvar*, *Task.getnumanz*, *Task.getobjsense*, *Task.getvarbound*, *Task.getvarboundslice*, *Task.getvarname*, *Task.getvarnameindex*, *Task.getvarnamelen*, *Task.inputdata*, *Task.putacollist*, *Task.putarowlist*, *Task.putatruncatetol*, *Task.putclist*, *Task.putconboundlist*, *Task.putconboundlistconst*, *Task.putconboundsliceconst*, *Task.putmaxnumanz*, *Task.putvarboundlist*, *Task.putvarboundlistconst*, *Task.putvarboundsliceconst*

Problem data - objective

- *Task.putbarcj* – Changes one element in barc.
- *Task.putcfix* – Replaces the fixed term in the objective.
- *Task.putcj* – Modifies one linear coefficient in the objective.
- *Task.putcslice* – Modifies a slice of the linear objective coefficients.
- *Task.putobjname* – Assigns a new name to the objective.
- *Task.putobjsense* – Sets the objective sense.
- *Task.putqobj* – Replaces all quadratic terms in the objective.
- *Task.putqobjij* – Replaces one coefficient in the quadratic term in the objective.
- *Infrequent:* *Task.putclist*

Problem data - quadratic part

- *Task.putqcon* – Replaces all quadratic terms in constraints.
- *Task.putqconk* – Replaces all quadratic terms in a single constraint.
- *Task.putqobj* – Replaces all quadratic terms in the objective.
- *Task.putqobjij* – Replaces one coefficient in the quadratic term in the objective.
- *Infrequent:* *Task.getmaxnumqnz*, *Task.getnumqconknz*, *Task.getnumqobjnz*, *Task.getqconk*, *Task.getqobj*, *Task.getqobjij*, *Task.putmaxnumqnz*
- *Deprecated:* ~~*Task.toconic*~~

Problem data - semidefinite

- *Task.appendbarvars* – Appends semidefinite variables to the problem.
- *Task.appendsparsesymmat* – Appends a general sparse symmetric matrix to the storage of symmetric matrices.
- *Task.appendsparsesymmatlist* – Appends a general sparse symmetric matrix to the storage of symmetric matrices.
- *Task.putafebarfentry* – Inputs one entry in barF.
- *Task.putafebarfentrylist* – Inputs a list of entries in barF.
- *Task.putafebarfrow* – Inputs a row of barF.
- *Task.putbaraij* – Inputs an element of barA.
- *Task.putbaraijlist* – Inputs list of elements of barA.
- *Task.putbararowlist* – Replace a set of rows of barA
- *Task.putbarcj* – Changes one element in barc.
- *Task.putbarvarname* – Sets the name of a semidefinite variable.
- *Infrequent:* *Task.emptyafebarfrow*, *Task.emptyafebarfrowlist*, *Task.getaccbarfblocktriplet*, *Task.getaccbarfnumblocktriplets*, *Task.getafebarfblocktriplet*, *Task.getafebarfnumblocktriplets*, *Task.getafebarfnumrowentries*, *Task.getafebarfrow*, *Task.getafebarfrowinfo*, *Task.getbarablocktriplet*, *Task.getbaraidx*, *Task.getbaraidxij*, *Task.getbaraidxinfo*, *Task.getbarasparsity*, *Task.getbarcblocktriplet*, *Task.getbarcidx*, *Task.getbarcidxinfo*, *Task.getbarcidxj*, *Task.getbarcsparsity*, *Task.getdimbarvarj*, *Task.getlenbarvarj*, *Task.getmaxnumbarvar*, *Task.getnumbarablocktriplets*, *Task.getnumbaranz*, *Task.getnumbarcblocktriplets*, *Task.getnumbarcnz*, *Task.getnumbarvar*, *Task.getnumsymmat*, *Task.getsparsesymmat*, *Task.getsymmatinfo*, *Task.putafebarfblocktriplet*, *Task.putbarablocktriplet*, *Task.putbarcblocktriplet*, *Task.putmaxnumbarvar*, *Task.removebarvars*

Problem data - variables

- *Task.appendvars* – Appends a number of variables to the optimization task.
- *Task.getnumvar* – Obtains the number of variables.
- *Task.putvarbound* – Changes the bounds for one variable.
- *Task.putvarboundslice* – Changes the bounds for a slice of the variables.
- *Task.putvarname* – Sets the name of a variable.

- *Task.putvartype* – Sets the variable type of one variable.
- *Task.removevars* – Removes a number of variables.
- *Infrequent:* *Task.chgvarbound*, *Task.generatebarvarnames*, *Task.generatevarnames*, *Task.getc*, *Task.getcj*, *Task.getmaxnumvar*, *Task.getnumintvar*, *Task.getvarbound*, *Task.getvarboundslice*, *Task.getvarname*, *Task.getvarnameindex*, *Task.getvarnamelen*, *Task.getvartype*, *Task.getvartypelist*, *Task.putclist*, *Task.putmaxnumvar*, *Task.putvarboundlist*, *Task.putvarboundlistconst*, *Task.putvarboundsliceconst*, *Task.putvartypelist*

Remote optimization

- *Task.asyncgetresult* – Request a solution from a remote job.
- *Task.asyncoptimize* – Offload the optimization task to a solver server in asynchronous mode.
- *Task.asyncpoll* – Requests information about the status of the remote job.
- *Task.asyncstop* – Request that the job identified by the token is terminated.
- *Task.optimizermt* – Offload the optimization task to a solver server and wait for the solution.
- *Task.putoptserverhost* – Specify an OptServer for remote calls.

Responses, errors and warnings

- *Env.getcodedesc* – Obtains a short description of a response code.

Sensitivity analysis

- *Task.dualsensitivity* – Performs sensitivity analysis on objective coefficients.
- *Task.primalsensitivity* – Perform sensitivity analysis on bounds.
- *Task.sensitivityreport* – Creates a sensitivity report.

Solution - dual

- *Task.getaccdoty* – Obtains the doty vector for an affine conic constraint.
- *Task.getdualobj* – Computes the dual objective value associated with the solution.
- *Task.gety* – Obtains the y vector for a solution.
- *Task.getyslice* – Obtains a slice of the y vector for a solution.
- *Infrequent:* *Task.getaccdotys*, *Task.getreducedcosts*, *Task.getslc*, *Task.getslcslice*, *Task.getslx*, *Task.getslxslice*, *Task.getsnx*, *Task.getsnxslice*, *Task.getsolution*, *Task.getsolutionnew*, *Task.getsolutionslice*, *Task.getsuc*, *Task.getsucslice*, *Task.getsux*, *Task.getsuxslice*, *Task.putaccdoty*, *Task.putconsolutioni*, *Task.putslc*, *Task.putslcslice*, *Task.putslx*, *Task.putslxslice*, *Task.putsnx*, *Task.putsnxslice*, *Task.putsolution*, *Task.putsolutionnew*, *Task.putsolutionyi*, *Task.putsuc*, *Task.putsucslice*, *Task.putsux*, *Task.putsuxslice*, *Task.putvarsolutionj*, *Task.putyslice*

Solution - primal

- *Task.getprimalobj* – Computes the primal objective value for the desired solution.
- *Task.getxx* – Obtains the xx vector for a solution.
- *Task.getxxslice* – Obtains a slice of the xx vector for a solution.
- *Task.putxx* – Sets the xx vector for a solution.
- *Task.putxxslice* – Sets a slice of the xx vector for a solution.
- *Infrequent:* *Task.evaluateacc*, *Task.evaluateaccs*, *Task.getsolution*, *Task.getsolutionnew*, *Task.getsolutionslice*, *Task.getxc*, *Task.getxcslice*, *Task.putconsolutioni*, *Task.putsolution*, *Task.putsolutionnew*, *Task.putvarsolutionj*, *Task.putxc*, *Task.putxcslice*, *Task.puty*

Solution - semidefinite

- *Task.getbarsj* – Obtains the dual solution for a semidefinite variable.
- *Task.getbarsslice* – Obtains the dual solution for a sequence of semidefinite variables.
- *Task.getbarxj* – Obtains the primal solution for a semidefinite variable.
- *Task.getbarxslice* – Obtains the primal solution for a sequence of semidefinite variables.
- *Infrequent:* *Task.putbarsj*, *Task.putbarxj*

Solution information

- *Task.getdualobj* – Computes the dual objective value associated with the solution.
- *Task.getprimalobj* – Computes the primal objective value for the desired solution.
- *Task.getprosta* – Obtains the problem status.
- *Task.getpviolcon* – Computes the violation of a primal solution associated to a constraint.
- *Task.getpviolvar* – Computes the violation of a primal solution for a list of scalar variables.
- *Task.getsolsta* – Obtains the solution status.
- *Task.getsolutioninfo* – Obtains information about of a solution.
- *Task.getsolutioninfonew* – Obtains information about of a solution.
- *Task.onesolutionsummary* – Prints a short summary of a specified solution.
- *Task.solutiondef* – Checks whether a solution is defined.
- *Task.solutionsummary* – Prints a short summary of the current solutions.
- *Infrequent:* *Task.analyzesolution*, *Task.deletesolution*, *Task.getdualsolutionnorms*, *Task.getdviolacc*, *Task.getdviolbarvar*, *Task.getdviolcon*, *Task.getdviolvar*, *Task.getprimalsolutionnorms*, *Task.getpviolacc*, *Task.getpviolbarvar*, *Task.getpvioldjc*, *Task.getskc*, *Task.getskcslice*, *Task.getskn*, *Task.getskx*, *Task.getskxslice*, *Task.getsolution*, *Task.getsolutionnew*, *Task.getsolutionslice*, *Task.putconsolutioni*, *Task.putskc*, *Task.putskcslice*, *Task.putskx*, *Task.putskxslice*, *Task.putsolution*, *Task.putsolutionnew*, *Task.putsolutionyi*, *Task.putvarsolutionj*
- *Deprecated:* ~~*Task.getdviolcones*~~, ~~*Task.getpviolcones*~~

Solving systems with basis matrix

- *Infrequent:* *Task.basiscond*, *Task.initbasissolve*, *Task.solvewithbasis*

System, memory and debugging

- *Infrequent:* *Task.checkmem*, *Task.getmemusage*

Versions

- *Env.getversion* – Obtains MOSEK version information.

15.3 Class Env

`mosek.Env`

The **MOSEK** global environment.

`Env.Env`

```
Env()
```

```
Env(licensefile=None, debugfile=None)
```

Constructor of a new environment.

Parameters

- `licensefile` (`str`) – License file to use. (input)
- `debugfile` (`str`) – File where the memory debugging log is written. (input)

Groups

Environment and task management

`Env.Task`

```
Task() -> task
```

```
Task(numcon, numvar) -> task
```

Creates a new task.

Parameters

- `numcon` (`int`) – An optional hint about the maximal number of constraints in the task. (input)
- `numvar` (`int`) – An optional hint about the maximal number of variables in the task. (input)

Return

`task` (*Task*) – A new task.

Groups

Environment and task management

`Env.__del__`

```
__del__()
```

Free the underlying native allocation.

Groups

Environment and task management

`Env.axpy`

```
axpy(n,alpha,x,y)
```

Adds αx to y, i.e. performs the update

$$y := \alpha x + y.$$

Note that the result is stored overwriting y. It must not overlap with the other input arrays.

Parameters

- `n` (`int32`) – Length of the vectors. (input)
- `alpha` (`float`) – The scalar that multiplies x. (input)
- `x` (`float[]`) – The x vector. (input)
- `y` (`float[]`) – The y vector. (input/output)

Groups

Linear algebra

`Env.checkinall`

```
checkinall()
```

Check in all unused license features to the license token server.

Groups

License system

`Env.checkinlicense`

```
checkinlicense(feature)
```

Check in a license feature to the license server. By default all licenses consumed by functions using a single environment are kept checked out for the lifetime of the **MOSEK** environment. This function checks in a given license feature back to the license server immediately.

If the given license feature is not checked out at all, or it is in use by a call to *Task.optimize*, calling this function has no effect.

Please note that returning a license to the license server incurs a small overhead, so frequent calls to this function should be avoided.

Parameters

`feature` (*mosek.feature*) – Feature to check in to the license system. (input)

Groups

License system

`Env.checkoutlicense`

```
checkoutlicense(feature)
```

Checks out a license feature from the license server. Normally the required license features will be automatically checked out the first time they are needed by the function *Task.optimize*. This function can be used to check out one or more features ahead of time.

The feature will remain checked out until the environment is deleted or the function *Env.checkinlicense* is called.

If a given feature is already checked out when this function is called, the call has no effect.

Parameters

`feature` (*mosek.feature*) – Feature to check out from the license system. (input)

Groups

License system

`Env.computesparsecholesky`

```
computesparsecholesky(numthreads,
                      ordermethod,
                      tolsingular,
                      anzc,
                      aptrc,
                      asubc,
                      avalc) ->
                     (perm,
                      diag,
                      lnzc,
                      lptrc,
                      lensubnval,
                      lsubc,
                      lvalc)
```

The function computes a Cholesky factorization of a sparse positive semidefinite matrix. Sparsity is exploited during the computations to reduce the amount of space and work required. Both the input and output matrices are represented using the sparse format.

To be precise, given a symmetric matrix $A \in \mathbb{R}^{n \times n}$ the function computes a nonsingular lower triangular matrix L, a diagonal matrix D and a permutation matrix P such that

$$LL^T - D = PAP^T.$$

If `ordermethod` is zero then reordering heuristics are not employed and P is the identity.

If a pivot during the computation of the Cholesky factorization is less than

$$-\rho \cdot \max((PAP^T)_{jj}, 1.0)$$

then the matrix is declared negative semidefinite. On the hand if a pivot is smaller than

$$\rho \cdot \max((PAP^T)_{jj}, 1.0),$$

then D_{jj} is increased from zero to

$$\rho \cdot \max((PAP^T)_{jj}, 1.0).$$

Therefore, if A is sufficiently positive definite then D will be the zero matrix. Here ρ is set equal to value of `tolsingular`.

Parameters

- `numthreads` (`int32`) – The number threads that can be used to do the computation. 0 means the code makes the choice. NOTE: API change in version 10: in versions up to 9 the argument in this position indicated whether to use multithreading or not. (input)
- `ordermethod` (`int32`) – If nonzero, then a sparsity preserving ordering will be employed. (input)
- `tolsingular` (`float`) – A positive parameter controlling when a pivot is declared zero. (input)
- `anzc` (`int32[]`) – `anzc[j]` is the number of nonzeros in the j-th column of A. (input)
- `aptrc` (`int64[]`) – `aptrc[j]` is a pointer to the first element in column j of A. (input)
- `asubc` (`int32[]`) – Row indexes for each column stored in increasing order. (input)
- `avalc` (`float[]`) – The value corresponding to row indexed stored in `asubc`. (input)

Return

- `perm` (`int32[]`) – Permutation array used to specify the permutation matrix P computed by the function.
- `diag` (`float[]`) – The diagonal elements of matrix D.
- `lnzc` (`int32[]`) – `lnzc[j]` is the number of non zero elements in column j of L.
- `lptrc` (`int64[]`) – `lptrc[j]` is a pointer to the first row index and value in column j of L.
- `lensubnval` (`int64`) – Number of elements in `lsubc` and `lvalc`.
- `lsubc` (`int32[]`) – Row indexes for each column stored in increasing order.
- `lvalc` (`float[]`) – The values corresponding to row indexed stored in `lsubc`.

Groups

Linear algebra

`Env.dot`

```
dot(n,x,y) -> (xty)
```

Computes the inner product of two vectors x, y of length $n \geq 0$, i.e

$$x \cdot y = \sum_{i=1}^{n} x_i y_i.$$

Note that if $n = 0$, then the result of the operation is 0.

Parameters

- `n` (`int32`) – Length of the vectors. (input)
- `x` (`float[]`) – The x vector. (input)
- `y` (`float[]`) – The y vector. (input)

Return

`xty` (`float`) – The result of the inner product between x and y.

Groups

Linear algebra

`Env.echointro`

```
echointro(longver)
```

Prints an intro to message stream.

Parameters

`longver` (`int32`) – If non-zero, then the intro is slightly longer. (input)

Groups

Logging

`Env.expirylicenses`

```
expirylicenses() -> (expiry)
```

Reports when the first license feature expires. It reports the number of days to the expiry of the first feature of all the features that were ever checked out from the start of the process, or from the last call to *`Env.resetexpirylicenses`*, until now.

Return

`expiry` (`int64`) – If nonnegative, then it is the minimum number days to expiry of any feature that has been checked out.

Groups

License system

`Env.gemm`

```
gemm(transa,transb,m,n,k,alpha,a,b,beta,c)
```

Performs a matrix multiplication plus addition of dense matrices. Given A, B and C of compatible dimensions, this function computes

$$C := \alpha op(A)op(B) + \beta C$$

where α, β are two scalar values. The function $op(X)$ denotes X if transX is *transpose.no*, or X^T if set to *transpose.yes*. The matrix C has m rows and n columns, and the other matrices must have compatible dimensions.

The result of this operation is stored in C. It must not overlap with the other input arrays.

Parameters

- `transa` (*mosek.transpose*) – Indicates whether the matrix A must be transposed. (input)
- `transb` (*mosek.transpose*) – Indicates whether the matrix B must be transposed. (input)
- `m` (`int32`) – Indicates the number of rows of matrix C. (input)
- `n` (`int32`) – Indicates the number of columns of matrix C. (input)
- `k` (`int32`) – Specifies the common dimension along which $op(A)$ and $op(B)$ are multiplied. For example, if neither A nor B are transposed, then this is the number of columns in A and also the number of rows in B. (input)
- `alpha` (`float`) – A scalar value multiplying the result of the matrix multiplication. (input)
- `a` (`float[]`) – The pointer to the array storing matrix A in a column-major format. (input)
- `b` (`float[]`) – The pointer to the array storing matrix B in a column-major format. (input)
- `beta` (`float`) – A scalar value that multiplies C. (input)
- `c` (`float[]`) – The pointer to the array storing matrix C in a column-major format. (input/output)

Groups

Linear algebra

`Env.gemv`

```
gemv(transa,m,n,alpha,a,x,beta,y)
```

Computes the multiplication of a scaled dense matrix times a dense vector, plus a scaled dense vector. Precisely, if `trans` is *transpose.no* then the update is

$$y := \alpha Ax + \beta y,$$

and if `trans` is *transpose.yes* then

$$y := \alpha A^T x + \beta y,$$

where α, β are scalar values and A is a matrix with m rows and n columns.

Note that the result is stored overwriting y. It must not overlap with the other input arrays.

Parameters

- `transa` (*mosek.transpose*) – Indicates whether the matrix A must be transposed. (input)
- `m` (`int32`) – Specifies the number of rows of the matrix A. (input)
- `n` (`int32`) – Specifies the number of columns of the matrix A. (input)
- `alpha` (`float`) – A scalar value multiplying the matrix A. (input)

- `a` (`float[]`) – A pointer to the array storing matrix A in a column-major format. (input)
- `x` (`float[]`) – A pointer to the array storing the vector x. (input)
- `beta` (`float`) – A scalar value multiplying the vector y. (input)
- `y` (`float[]`) – A pointer to the array storing the vector y. (input/output)

Groups

Linear algebra

`Env.getcodedesc`

```
@staticmethod
getcodedesc(code) -> (symname,str)
```

Obtains a short description of the meaning of the response code given by `code`.

Parameters

`code` (*mosek.rescode*) – A valid **MOSEK** response code. (input)

Return

- `symname` (`str`) – Symbolic name corresponding to `code`.
- `str` (`str`) – Obtains a short description of a response code.

Groups

Names, *Responses, errors and warnings*

`Env.getversion`

```
@staticmethod
getversion() -> (major,minor,revision)
```

Obtains **MOSEK** version information.

Return

- `major` (`int32`) – Major version number.
- `minor` (`int32`) – Minor version number.
- `revision` (`int32`) – Revision number.

Groups

Versions

`Env.licensecleanup`

```
@staticmethod
licensecleanup()
```

Stops all threads and deletes all handles used by the license system. If this function is called, it must be called as the last **MOSEK** API call. No other **MOSEK** API calls are valid after this.

Groups

License system

`Env.linkfiletostream`

```
linkfiletostream(whichstream,filename,append)
```

Sends all output from the stream defined by `whichstream` to the file given by `filename`.

Parameters

- `whichstream` (*mosek.streamtype*) – Index of the stream. (input)
- `filename` (`str`) – A valid file name. (input)
- `append` (`int32`) – If this argument is 0 the file will be overwritten, otherwise it will be appended to. (input)

Groups

Logging

`Env.optimizebatch`

```
optimizebatch(israce,
              maxtime,
              numthreads,
              task,
              trmcode,
              rcode)
```

```
optimizebatch(israce,maxtime,numthreads,task) -> (trmcode,rcode)
```

Optimize a number of tasks in parallel using a specified number of threads. All callbacks and log output streams are disabled.

Assuming that each task takes about same time and there many more tasks than number of threads then a linear speedup can be achieved, also known as strong scaling. A typical application of this method is to solve many small tasks of similar type; in this case it is recommended that each of them is allocated a single thread by setting *iparam.num_threads* to 1.

If the parameters `israce` or `maxtime` are used, then the result may not be deterministic, in the sense that the tasks which complete first may vary between runs.

The remaining behavior, including termination and response codes returned for each task, are the same as if each task was optimized separately.

Parameters

- `israce` (`bool`) – If nonzero, then the function is terminated after the first task has been completed. (input)
- `maxtime` (`float`) – Time limit for the function: if nonnegative, then the function is terminated after maxtime (seconds) has expired. (input)
- `numthreads` (`int32`) – Number of threads to be employed. (input)
- `task` (*Task*`[]`) – An array of tasks to optimize in parallel. (input)
- `trmcode` (*mosek.rescode*`[]`) – The termination code for each task. (output)
- `rcode` (*mosek.rescode*`[]`) – The response code for each task. (output)

Return

- `trmcode` (*mosek.rescode*`[]`) – The termination code for each task.
- `rcode` (*mosek.rescode*`[]`) – The response code for each task.

Groups

Optimization

`Env.potrf`

```
potrf(uplo,n,a)
```

Computes a Cholesky factorization of a real symmetric positive definite dense matrix.

Parameters

- `uplo` (*mosek.uplo*) – Indicates whether the upper or lower triangular part of the matrix is stored. (input)
- `n` (`int32`) – Dimension of the symmetric matrix. (input)
- `a` (`float[]`) – A symmetric matrix stored in column-major order. Only the lower or the upper triangular part is used, accordingly with the `uplo` parameter. It will contain the result on exit. (input/output)

Groups

Linear algebra

`Env.putlicensecode`

```
putlicensecode(code)
```

Input a runtime license code. This function has an effect only before the first optimization.

Parameters
`code` (`int32[]`) – A runtime license code. (input)

Groups
License system

`Env.putlicensedebug`

```
putlicensedebug(licdebug)
```

Enables debug information for the license system. If `licdebug` is non-zero, then **MOSEK** will print debug info regarding the license checkout.

Parameters
`licdebug` (`int32`) – Whether license checkout debug info should be printed. (input)

Groups
License system

`Env.putlicensepath`

```
putlicensepath(licensepath)
```

Set the path to the license file. This function has an effect only before the first optimization.

Parameters
`licensepath` (`str`) – A path specifying where to search for the license. (input)

Groups
License system

`Env.putlicensewait`

```
putlicensewait(licwait)
```

Control whether **MOSEK** should wait for an available license if no license is available. If `licwait` is non-zero, then **MOSEK** will wait for `licwait-1` milliseconds between each check for an available license.

Parameters
`licwait` (`int32`) – Whether **MOSEK** should wait for a license if no license is available. (input)

Groups
License system

`Env.resetexpirylicenses`

```
resetexpirylicenses()
```

Reset the license expiry reporting startpoint.

Groups
License system

`Env.set_Stream`

```
set_Stream(whichstream, callback)
```

Directs all output from a environment stream to a callback function.

Parameters

- `whichstream` (*streamtype*) – Index of the stream. (input)
- `callback` (*streamfunc*) – The callback function. (input)

Groups

Logging, *Callback*

`Env.sparsetriangularsolvedense`

```
sparsetriangularsolvedense(transposed,lnzc,lptrc,lsubc,lvalc,b)
```

The function solves a triangular system of the form

$$Lx = b$$

or

$$L^T x = b$$

where L is a sparse lower triangular nonsingular matrix. This implies in particular that diagonals in L are nonzero.

Parameters

- `transposed` (*mosek.transpose*) – Controls whether to use with L or L^T. (input)
- `lnzc` (`int32[]`) – `lnzc[j]` is the number of nonzeros in column `j`. (input)
- `lptrc` (`int64[]`) – `lptrc[j]` is a pointer to the first row index and value in column `j`. (input)
- `lsubc` (`int32[]`) – Row indexes for each column stored sequentially. Must be stored in increasing order for each column. (input)
- `lvalc` (`float[]`) – The value corresponding to the row index stored in `lsubc`. (input)
- `b` (`float[]`) – The right-hand side of linear equation system to be solved as a dense vector. (input/output)

Groups

Linear algebra

`Env.syeig`

```
syeig(uplo,n,a,w)
```

```
syeig(uplo,n,a) -> (w)
```

Computes all eigenvalues of a real symmetric matrix A. Given a matrix $A \in \mathbb{R}^{n \times n}$ it returns a vector $w \in \mathbb{R}^n$ containing the eigenvalues of A.

Parameters

- `uplo` (*mosek.uplo*) – Indicates whether the upper or lower triangular part is used. (input)
- `n` (`int32`) – Dimension of the symmetric input matrix. (input)
- `a` (`float[]`) – A symmetric matrix A stored in column-major order. Only the part indicated by `uplo` is used. (input)
- `w` (`float[]`) – Array of length at least `n` containing the eigenvalues of A. (output)

Return

`w` (`float[]`) – Array of length at least `n` containing the eigenvalues of A.

Groups

Linear algebra

`Env.syevd`

```
syevd(uplo,n,a,w)
```

```
syevd(uplo,n,a) -> (w)
```

Computes all the eigenvalues and eigenvectors a real symmetric matrix. Given the input matrix $A \in \mathbb{R}^{n \times n}$, this function returns a vector $w \in \mathbb{R}^n$ containing the eigenvalues of A and it also computes the eigenvectors of A. Therefore, this function computes the eigenvalue decomposition of A as

$$A = UVU^T,$$

where $V = \mathbf{diag}(w)$ and U contains the eigenvectors of A.

Note that the matrix U overwrites the input data A.

Parameters

- `uplo` (*mosek.uplo*) – Indicates whether the upper or lower triangular part is used. (input)
- `n` (`int32`) – Dimension of the symmetric input matrix. (input)
- `a` (`float[]`) – A symmetric matrix A stored in column-major order. Only the part indicated by `uplo` is used. On exit it will be overwritten by the matrix U. (input/output)
- `w` (`float[]`) – Array of length at least `n` containing the eigenvalues of A. (output)

Return

`w` (`float[]`) – Array of length at least `n` containing the eigenvalues of A.

Groups

Linear algebra

`Env.syrk`

```
syrk(uplo,trans,n,k,alpha,a,beta,c)
```

Performs a symmetric rank-k update for a symmetric matrix.

Given a symmetric matrix $C \in \mathbb{R}^{n \times n}$, two scalars α, β and a matrix A of rank $k \leq n$, it computes either

$$C := \alpha AA^T + \beta C,$$

when `trans` is set to *transpose.no* and $A \in \mathbb{R}^{n \times k}$, or

$$C := \alpha A^T A + \beta C,$$

when `trans` is set to *transpose.yes* and $A \in \mathbb{R}^{k \times n}$.

Only the part of C indicated by `uplo` is used and only that part is updated with the result. It must not overlap with the other input arrays.

Parameters

- `uplo` (*mosek.uplo*) – Indicates whether the upper or lower triangular part of C is used. (input)
- `trans` (*mosek.transpose*) – Indicates whether the matrix A must be transposed. (input)
- `n` (`int32`) – Specifies the order of C. (input)
- `k` (`int32`) – Indicates the number of rows or columns of A, depending on whether or not it is transposed, and its rank. (input)
- `alpha` (`float`) – A scalar value multiplying the result of the matrix multiplication. (input)
- `a` (`float[]`) – The pointer to the array storing matrix A in a column-major format. (input)

- `beta` (`float`) – A scalar value that multiplies C. (input)
- `c` (`float[]`) – The pointer to the array storing matrix C in a column-major format. (input/output)

Groups

Linear algebra

15.4 Class Task

`mosek.Task`

Represents an optimization task.

`Task.Task`

```
Task()
```

```
Task(env)
```

```
Task(env, numcon, numvar)
```

```
Task(task)
```

Constructor of a new optimization task.

Parameters

- `env` (*Env*) – Parent environment. (input)
- `numcon` (`int`) – An optional hint about the maximal number of constraints in the task. (input)
- `numvar` (`int`) – An optional hint about the maximal number of variables in the task. (input)
- `task` (*Task*) – A task that will be cloned. (input)

Groups

Environment and task management

`Task.__del__`

```
__del__()
```

Free the underlying native allocation.

Groups

Environment and task management

`Task.analyzenames`

```
analyzenames(whichstream,nametype)
```

The function analyzes the names and issues an error if a name is invalid.

Parameters

- `whichstream` (*mosek.streamtype*) – Index of the stream. (input)
- `nametype` (*mosek.nametype*) – The type of names e.g. valid in MPS or LP files. (input)

Groups

Names

`Task.analyzeproblem`

```
analyzeproblem(whichstream)
```

The function analyzes the data of a task and writes out a report.

Parameters

`whichstream` (*mosek.streamtype*) – Index of the stream. (input)

Groups

Inspecting the task

`Task.analyzesolution`

```
analyzesolution(whichstream,whichsol)
```

Print information related to the quality of the solution and other solution statistics.

By default this function prints information about the largest infeasibilites in the solution, the primal (and possibly dual) objective value and the solution status.

Following parameters can be used to configure the printed statistics:

- *iparam.ana_sol_basis* enables or disables printing of statistics specific to the basis solution (condition number, number of basic variables etc.). Default is on.
- *iparam.ana_sol_print_violated* enables or disables listing names of all constraints (both primal and dual) which are violated by the solution. Default is off.
- *dparam.ana_sol_infeas_tol* is the tolerance defining when a constraint is considered violated. If a constraint is violated more than this, it will be listed in the summary.

Parameters

- `whichstream` (*mosek.streamtype*) – Index of the stream. (input)
- `whichsol` (*mosek.soltype*) – Selects a solution. (input)

Groups

Solution information, *Inspecting the task*

`Task.appendacc`

```
appendacc(domidx,afeidxlist,b)
```

Appends an affine conic constraint to the task. The affine constraint has the form *a sequence of affine expressions belongs to a domain.*

The domain index is specified with `domidx` and should refer to a domain previously appended with one of the `append...domain` functions.

The length of the affine expression list `afeidxlist` must be equal to the dimension n of the domain. The elements of `afeidxlist` are indexes to the store of affine expressions, i.e. the affine expressions appearing in the affine conic constraint are:

$$F_{\texttt{afeidxlist}[k],:}x + g_{\texttt{afeidxlist}[k]} \quad \text{for } k = 0, \ldots, n-1.$$

If an optional vector `b` of the same length as `afeidxlist` is specified then the expressions appearing in the affine constraint will instead be taken as:

$$F_{\texttt{afeidxlist}[k],:}x + g_{\texttt{afeidxlist}[k]} - b_k \quad \text{for } k = 0, \ldots, n-1.$$

Parameters

- `domidx` (`int64`) – Domain index. (input)
- `afeidxlist` (`int64[]`) – List of affine expression indexes. (input)
- `b` (`float[]`) – The vector of constant terms modifying affine expressions. Optional, can be `None` if not required. (input)

Groups

Problem data - affine conic constraints

`Task.appendaccs`

```
appendaccs(domidxs,afeidxlist,b)
```

Appends `numaccs` affine conic constraint to the task. Each single affine conic constraint should be specified as in *Task.appendacc* and the input of this function should contain the concatenation of all these descriptions.

In particular, the length of `afeidxlist` must equal the sum of dimensions of domains indexed in `domainsidxs`.

Parameters

- `domidxs` (`int64[]`) – Domain indices. (input)
- `afeidxlist` (`int64[]`) – List of affine expression indexes. (input)
- `b` (`float[]`) – The vector of constant terms modifying affine expressions. Optional, can be `None` if not required. (input)

Groups

Problem data - affine conic constraints

`Task.appendaccseq`

```
appendaccseq(domidx,afeidxfirst,b)
```

Appends an affine conic constraint to the task, as in *Task.appendacc*. The function assumes the affine expressions forming the constraint are sequential. The affine constraint has the form *a sequence of affine expressions belongs to a domain*.

The domain index is specified with `domidx` and should refer to a domain previously appended with one of the `append...domain` functions.

The number of affine expressions should be equal to the dimension n of the domain. The affine expressions forming the affine constraint are arranged sequentially in a contiguous block of the affine expression store starting from position `afeidxfirst`. That is, the affine expressions appearing in the affine conic constraint are:

$$F_{\texttt{afeidxfirst}+k,:}x + g_{\texttt{afeidxfirst}+k} \quad \text{for } k = 0, \ldots, n-1.$$

If an optional vector `b` of length `numafeidx` is specified then the expressions appearing in the affine constraint will instead be taken as

$$F_{\texttt{afeidxfirst}+k,:}x + g_{\texttt{afeidxfirst}+k} - b_k \quad \text{for } k = 0, \ldots, n-1.$$

Parameters

- `domidx` (`int64`) – Domain index. (input)
- `afeidxfirst` (`int64`) – Index of the first affine expression. (input)
- `b` (`float[]`) – The vector of constant terms modifying affine expressions. Optional, can be `None` if not required. (input)

Groups

Problem data - affine conic constraints

`Task.appendaccsseq`

```
appendaccsseq(domidxs,numafeidx,afeidxfirst,b)
```

Appends `numaccs` affine conic constraint to the task. It is the block variant of *Task.appendaccs*, that is it assumes that the affine expressions appearing in the affine conic constraints are sequential in the affine expression store, starting from position `afeidxfirst`.

Parameters

- `domidxs` (`int64[]`) – Domain indices. (input)

- `numafeidx` (`int64`) – Number of affine expressions in the affine expression list (must equal the sum of dimensions of the domains). (input)
- `afeidxfirst` (`int64`) – Index of the first affine expression. (input)
- `b` (`float[]`) – The vector of constant terms modifying affine expressions. Optional, can be `None` if not required. (input)

Groups
Problem data - affine conic constraints

`Task.appendafes`

```
appendafes(num)
```

Appends a number of empty affine expressions to the task.

Parameters
`num` (`int64`) – Number of empty affine expressions which should be appended. (input)

Groups
Problem data - affine expressions

`Task.appendbarvars`

```
appendbarvars(dim)
```

Appends positive semidefinite matrix variables of dimensions given by `dim` to the problem.

Parameters
`dim` (`int32[]`) – Dimensions of symmetric matrix variables to be added. (input)

Groups
Problem data - semidefinite

~~`Task.appendcone`~~ *Deprecated*

```
appendcone(ct,conepar,submem)
```

NOTE: This interface to conic optimization is deprecated and will be removed in a future major release. Conic problems should be specified using the affine conic constraints interface (ACC), see Sec. 6.2 for details.

Appends a new conic constraint to the problem. Hence, add a constraint

$$\hat{x} \in \mathcal{K}$$

to the problem, where $\mathcal{K}$ is a convex cone. $\hat{x}$ is a subset of the variables which will be specified by the argument `submem`. Cone type is specified by `ct`.

Define

$$\hat{x} = x_{\texttt{submem}[1]}, \ldots, x_{\texttt{submem}[\texttt{nummem}]}.$$

Depending on the value of `ct` this function appends one of the constraints:

- Quadratic cone (*`conetype.quad`*, requires $\texttt{nummem} \geq 1$):

$$\hat{x}_0 \geq \sqrt{\sum_{i=1}^{i<\texttt{nummem}} \hat{x}_i^2}$$

- Rotated quadratic cone (*`conetype.rquad`*, requires $\texttt{nummem} \geq 2$):

$$2\hat{x}_0\hat{x}_1 \geq \sum_{i=2}^{i<\texttt{nummem}} \hat{x}_i^2, \quad \hat{x}_0, \hat{x}_1 \geq 0$$

- Primal exponential cone (*conetype.pexp*, requires `nummem` $= 3$):

$$\hat{x}_0 \geq \hat{x}_1 \exp(\hat{x}_2/\hat{x}_1), \quad \hat{x}_0, \hat{x}_1 \geq 0$$

- Primal power cone (*conetype.ppow*, requires `nummem` ≥ 2):

$$\hat{x}_0^{\alpha} \hat{x}_1^{1-\alpha} \geq \sqrt{\sum_{i=2}^{i<\texttt{nummem}} \hat{x}_i^2}, \quad \hat{x}_0, \hat{x}_1 \geq 0$$

  where α is the cone parameter specified by `conepar`.

- Dual exponential cone (*conetype.dexp*, requires `nummem` $= 3$):

$$\hat{x}_0 \geq -\hat{x}_2 e^{-1} \exp(\hat{x}_1/\hat{x}_2), \quad \hat{x}_2 \leq 0, \hat{x}_0 \geq 0$$

- Dual power cone (*conetype.dpow*, requires `nummem` ≥ 2):

$$\left(\frac{\hat{x}_0}{\alpha}\right)^{\alpha} \left(\frac{\hat{x}_1}{1-\alpha}\right)^{1-\alpha} \geq \sqrt{\sum_{i=2}^{i<\texttt{nummem}} \hat{x}_i^2}, \quad \hat{x}_0, \hat{x}_1 \geq 0$$

  where α is the cone parameter specified by `conepar`.

- Zero cone (*conetype.zero*):

$$\hat{x}_i = 0 \text{ for all } i$$

Please note that the sets of variables appearing in different conic constraints must be disjoint.

For an explained code example see Sec. 6.3, Sec. 6.5 or Sec. 6.4.

Parameters

- `ct` (*mosek.conetype*) – Specifies the type of the cone. (input)
- `conepar` (`float`) – For the power cone it denotes the exponent alpha. For other cone types it is unused and can be set to 0. (input)
- `submem` (`int32[]`) – Variable subscripts of the members in the cone. (input)

Groups

Problem data - cones (deprecated)

~~`Task.appendconeseq`~~ *Deprecated*

```
appendconeseq(ct,conepar,nummem,j)
```

NOTE: This interface to conic optimization is deprecated and will be removed in a future major release. Conic problems should be specified using the affine conic constraints interface (ACC), see Sec. 6.2 for details.

Appends a new conic constraint to the problem, as in ~~*Task.appendcone*~~. The function assumes the members of cone are sequential where the first member has index `j` and the last `j+nummem-1`.

Parameters

- `ct` (*mosek.conetype*) – Specifies the type of the cone. (input)
- `conepar` (`float`) – For the power cone it denotes the exponent alpha. For other cone types it is unused and can be set to 0. (input)
- `nummem` (`int32`) – Number of member variables in the cone. (input)
- `j` (`int32`) – Index of the first variable in the conic constraint. (input)

Groups

Problem data - cones (deprecated)

~~`Task.appendconesseq`~~ *Deprecated*

```
appendconesseq(ct,conepar,nummem,j)
```

NOTE: This interface to conic optimization is deprecated and will be removed in a future major release. Conic problems should be specified using the affine conic constraints interface (ACC), see Sec. 6.2 for details.

Appends a number of conic constraints to the problem, as in ~~*Task.appendcone*~~. The kth cone is assumed to be of dimension `nummem[k]`. Moreover, it is assumed that the first variable of the first cone has index j and starting from there the sequentially following variables belong to the first cone, then to the second cone and so on.

Parameters

- `ct` (*mosek.conetype* `[]`) – Specifies the type of the cone. (input)
- `conepar` (`float[]`) – For the power cone it denotes the exponent alpha. For other cone types it is unused and can be set to 0. (input)
- `nummem` (`int32[]`) – Numbers of member variables in the cones. (input)
- `j` (`int32`) – Index of the first variable in the first cone to be appended. (input)

Groups

Problem data - cones (deprecated)

`Task.appendcons`

```
appendcons(num)
```

Appends a number of constraints to the model. Appended constraints will be declared free. Please note that **MOSEK** will automatically expand the problem dimension to accommodate the additional constraints.

Parameters

`num` (`int32`) – Number of constraints which should be appended. (input)

Groups

Problem data - linear part, *Problem data - constraints*

`Task.appenddjcs`

```
appenddjcs(num)
```

Appends a number of empty disjunctive constraints to the task.

Parameters

`num` (`int64`) – Number of empty disjunctive constraints which should be appended. (input)

Groups

Problem data - disjunctive constraints

`Task.appenddualexpconedomain`

```
appenddualexpconedomain() -> (domidx)
```

Appends the dual exponential cone $\left\{x \in \mathbb{R}^3 \ : \ x_0 \geq -x_2 e^{-1} e^{x_1/x_2}, \ x_0 > 0, \ x_2 < 0\right\}$ to the list of domains.

Return

`domidx` (`int64`) – Index of the domain.

Groups

Problem data - domain

`Task.appenddualgeomeanconedomain`

```
appenddualgeomeanconedomain(n) -> (domidx)
```

Appends the dual geometric mean cone $\left\{x \in \mathbb{R}^n \ : \ (n-1)\left(\prod_{i=0}^{n-2} x_i\right)^{1/(n-1)} \geq |x_{n-1}|, \ x_0, \ldots, x_{n-2} \geq 0\right\}$ to the list of domains.

Parameters
`n` (`int64`) – Dimension of the domain. (input)

Return
`domidx` (`int64`) – Index of the domain.

Groups
Problem data - domain

`Task.appenddualpowerconedomain`

```
appenddualpowerconedomain(n,alpha) -> (domidx)
```

Appends the dual power cone domain of dimension n, with n_ℓ variables appearing on the left-hand side, where n_ℓ is the length of α, and with a homogenous sequence of exponents $\alpha_0, \ldots, \alpha_{n_\ell - 1}$.

Formally, let $s = \sum_i \alpha_i$ and $\beta_i = \alpha_i / s$, so that $\sum_i \beta_i = 1$. Then the dual power cone is defined as follows:

$$\left\{x \in \mathbb{R}^n \ : \ \prod_{i=0}^{n_\ell - 1} \left(\frac{x_i}{\beta_i}\right)^{\beta_i} \geq \sqrt{\sum_{j=n_\ell}^{n-1} x_j^2}, \ x_0 \ldots, x_{n_\ell - 1} \geq 0\right\}$$

Parameters
- `n` (`int64`) – Dimension of the domain. (input)
- `alpha` (`float[]`) – The sequence proportional to exponents. Must be positive. (input)

Return
`domidx` (`int64`) – Index of the domain.

Groups
Problem data - domain

`Task.appenddualpowerconedomainseq`

```
appenddualpowerconedomainseq(n,nleft,alpha,domidxlist)
```

```
appenddualpowerconedomainseq(n,nleft,alpha) -> (domidxlist)
```

Appends a sequence of dual power cone domains. The input arrays contain concatenated descriptions of individual domains, where each domain is defined as in *Task.appenddualpowerconedomain*.

Parameters
- `n` (`int64[]`) – Dimensions of the domains. (input)
- `nleft` (`int64[]`) – Number of variables on the left hand sides. (input)
- `alpha` (`float[]`) – The sequences proportional to exponents, concatenated for all domains. Must be positive. (input)
- `domidxlist` (`int64[]`) – Indexes of the domains. (output)

Return
`domidxlist` (`int64[]`) – Indexes of the domains.

Groups
Problem data - domain

`Task.appendprimalexpconedomain`

```
appendprimalexpconedomain() -> (domidx)
```

Appends the primal exponential cone $\left\{x \in \mathbb{R}^3 \ : \ x_0 \geq x_1 e^{x_2/x_1}, \ x_0, x_1 > 0\right\}$ to the list of domains.

Return
`domidx` (`int64`) – Index of the domain.

Groups
Problem data - domain

`Task.appendprimalgeomeanconedomain`

```
appendprimalgeomeanconedomain(n) -> (domidx)
```

Appends the primal geometric mean cone $\left\{x \in \mathbb{R}^n \ : \ \left(\prod_{i=0}^{n-2} x_i\right)^{1/(n-1)} \geq |x_{n-1}|, \ x_0 \ldots, x_{n-2} \geq 0\right\}$ to the list of domains.

Parameters
`n` (`int64`) – Dimension of the domain. (input)

Return
`domidx` (`int64`) – Index of the domain.

Groups
Problem data - domain

`Task.appendprimalpowerconedomain`

```
appendprimalpowerconedomain(n,alpha) -> (domidx)
```

Appends the primal power cone domain of dimension n, with n_ℓ variables appearing on the left-hand side, where n_ℓ is the length of α, and with a homogenous sequence of exponents $\alpha_0, \ldots, \alpha_{n_\ell - 1}$.

Formally, let $s = \sum_i \alpha_i$ and $\beta_i = \alpha_i / s$, so that $\sum_i \beta_i = 1$. Then the primal power cone is defined as follows:

$$\left\{x \in \mathbb{R}^n \ : \ \prod_{i=0}^{n_\ell - 1} x_i^{\beta_i} \geq \sqrt{\sum_{j=n_\ell}^{n-1} x_j^2}, \ x_0 \ldots, x_{n_\ell - 1} \geq 0\right\}$$

Parameters
- `n` (`int64`) – Dimension of the domain. (input)
- `alpha` (`float[]`) – The sequence proportional to exponents. Must be positive. (input)

Return
`domidx` (`int64`) – Index of the domain.

Groups
Problem data - domain

`Task.appendprimalpowerconedomainseq`

```
appendprimalpowerconedomainseq(n,nleft,alpha,domidxlist)
```

```
appendprimalpowerconedomainseq(n,nleft,alpha) -> (domidxlist)
```

Appends a sequence of primal power cone domains. The input arrays contain concatenated descriptions of individual domains, where each domain is defined as in *Task.appendprimalpowerconedomain*.

Parameters
- `n` (`int64[]`) – Dimensions of the domains. (input)
- `nleft` (`int64[]`) – Number of variables on the left hand sides. (input)

- `alpha` (`float[]`) – The sequences proportional to exponents, concatenated for all domains. Must be positive. (input)
- `domidxlist` (`int64[]`) – Indexes of the domains. (output)

Return
`domidxlist` (`int64[]`) – Indexes of the domains.

Groups
Problem data - domain

`Task.appendquadraticconedomain`

```
appendquadraticconedomain(n) -> (domidx)
```

Appends the n-dimensional quadratic cone $\left\{x \in \mathbb{R}^n \ : \ x_0 \geq \sqrt{\sum_{i=1}^{n-1} x_i^2}\right\}$ to the list of domains.

Parameters
`n` (`int64`) – Dimension of the domain. (input)

Return
`domidx` (`int64`) – Index of the domain.

Groups
Problem data - domain

`Task.appendrdomain`

```
appendrdomain(n) -> (domidx)
```

Appends the n-dimensional real space $\{x \in \mathbb{R}^n\}$ to the list of domains.

Parameters
`n` (`int64`) – Dimension of the domain. (input)

Return
`domidx` (`int64`) – Index of the domain.

Groups
Problem data - domain

`Task.appendrminusdomain`

```
appendrminusdomain(n) -> (domidx)
```

Appends the n-dimensional negative orthant $\{x \in \mathbb{R}^n : x \leq 0\}$ to the list of domains.

Parameters
`n` (`int64`) – Dimension of the domain. (input)

Return
`domidx` (`int64`) – Index of the domain.

Groups
Problem data - domain

`Task.appendrplusdomain`

```
appendrplusdomain(n) -> (domidx)
```

Appends the n-dimensional positive orthant $\{x \in \mathbb{R}^n : x \geq 0\}$ to the list of domains.

Parameters
`n` (`int64`) – Dimension of the domain. (input)

Return
`domidx` (`int64`) – Index of the domain.

Groups
Problem data - domain

`Task.appendrquadraticconedomain`

```
appendrquadraticconedomain(n) -> (domidx)
```

Appends the n-dimensional rotated quadratic cone $\left\{x \in \mathbb{R}^n \ : \ 2x_0x_1 \geq \sum_{i=2}^{n-1} x_i^2, \ x_0, x_1 \geq 0\right\}$ to the list of domains.

Parameters
`n` (`int64`) – Dimension of the domain. (input)

Return
`domidx` (`int64`) – Index of the domain.

Groups
Problem data - domain

`Task.appendrzerodomain`

```
appendrzerodomain(n) -> (domidx)
```

Appends the zero in n-dimensional real space $\{x \in \mathbb{R}^n : x = 0\}$ to the list of domains.

Parameters
`n` (`int64`) – Dimension of the domain. (input)

Return
`domidx` (`int64`) – Index of the domain.

Groups
Problem data - domain

`Task.appendsparsesymmat`

```
appendsparsesymmat(dim,subi,subj,valij) -> (idx)
```

MOSEK maintains a storage of symmetric data matrices that is used to build $\overline{C}$ and $\overline{A}$. The storage can be thought of as a vector of symmetric matrices denoted E. Hence, E_i is a symmetric matrix of certain dimension.

This function appends a general sparse symmetric matrix on triplet form to the vector E of symmetric matrices. The vectors `subi`, `subj`, and `valij` contains the row subscripts, column subscripts and values of each element in the symmetric matrix to be appended. Since the matrix that is appended is symmetric, only the lower triangular part should be specified. Moreover, duplicates are not allowed.

Observe the function reports the index (position) of the appended matrix in E. This index should be used for later references to the appended matrix.

Parameters
- `dim` (`int32`) – Dimension of the symmetric matrix that is appended. (input)
- `subi` (`int32[]`) – Row subscript in the triplets. (input)
- `subj` (`int32[]`) – Column subscripts in the triplets. (input)
- `valij` (`float[]`) – Values of each triplet. (input)

Return
`idx` (`int64`) – Unique index assigned to the inputted matrix that can be used for later reference.

Groups
Problem data - semidefinite

`Task.appendsparsesymmatlist`

```
appendsparsesymmatlist(dims,nz,subi,subj,valij,idx)
```

```
appendsparsesymmatlist(dims,nz,subi,subj,valij) -> (idx)
```

MOSEK maintains a storage of symmetric data matrices that is used to build $\overline{C}$ and $\overline{A}$. The storage can be thought of as a vector of symmetric matrices denoted E. Hence, E_i is a symmetric matrix of certain dimension.

This function appends general sparse symmetric matrixes on triplet form to the vector E of symmetric matrices. The vectors `subi`, `subj`, and `valij` contains the row subscripts, column subscripts and values of each element in the symmetric matrix to be appended. Since the matrix that is appended is symmetric, only the lower triangular part should be specified. Moreover, duplicates are not allowed.

Observe the function reports the index (position) of the appended matrix in E. This index should be used for later references to the appended matrix.

Parameters

- `dims` (`int32[]`) – Dimensions of the symmetric matrixes. (input)
- `nz` (`int64[]`) – Number of nonzeros for each matrix. (input)
- `subi` (`int32[]`) – Row subscript in the triplets. (input)
- `subj` (`int32[]`) – Column subscripts in the triplets. (input)
- `valij` (`float[]`) – Values of each triplet. (input)
- `idx` (`int64[]`) – Unique index assigned to the inputted matrix that can be used for later reference. (output)

Return

`idx` (`int64[]`) – Unique index assigned to the inputted matrix that can be used for later reference.

Groups

Problem data - semidefinite

`Task.appendsvecpsdconedomain`

```
appendsvecpsdconedomain(n) -> (domidx)
```

Appends the domain consisting of vectors of length $n = d(d+1)/2$ defined as follows

$$\{(x_1, \ldots, x_{d(d+1)/2}) \in \mathbb{R}^n \ : \ \mathrm{sMat}(x) \in \mathcal{S}_+^d\} = \{\mathrm{sVec}(X) \ : \ X \in \mathcal{S}_+^d\},$$

where

$$\mathrm{sVec}(X) = (X_{11}, \sqrt{2}X_{21}, \ldots, \sqrt{2}X_{d1}, X_{22}, \sqrt{2}X_{32}, \ldots, X_{dd}),$$

and

$$\mathrm{sMat}(x) = \begin{bmatrix} x_1 & x_2/\sqrt{2} & \cdots & x_d/\sqrt{2} \\ x_2/\sqrt{2} & x_{d+1} & \cdots & x_{2d-1}/\sqrt{2} \\ \cdots & \cdots & \cdots & \cdots \\ x_d/\sqrt{2} & x_{2d-1}/\sqrt{2} & \cdots & x_{d(d+1)/2} \end{bmatrix}.$$

In other words, the domain consists of vectorizations of the lower-triangular part of a positive semidefinite matrix, with the non-diagonal elements additionally rescaled.

This domain is a self-dual cone.

Parameters

`n` (`int64`) – Dimension of the domain, must be of the form $d(d+1)/2$. (input)

Return

`domidx` (`int64`) – Index of the domain.

Groups

Problem data - domain

`Task.appendvars`

```
appendvars(num)
```

Appends a number of variables to the model. Appended variables will be fixed at zero. Please note that **MOSEK** will automatically expand the problem dimension to accommodate the additional variables.

Parameters

`num` (`int32`) – Number of variables which should be appended. (input)

Groups

Problem data - linear part, *Problem data - variables*

`Task.asyncgetresult`

```
asyncgetresult(address,accesstoken,token) -> (respavailable,resp,trm)
```

Request a solution from a remote job identified by the argument `token`. For other arguments see *Task.asyncoptimize*. If the solution is available it will be retrieved and loaded into the local task.

Parameters

- `address` (`str`) – Address of the OptServer. (input)
- `accesstoken` (`str`) – Access token. (input)
- `token` (`str`) – The task token. (input)

Return

- `respavailable` (`bool`) – Indicates if a remote response is available. If this is not true, `resp` and `trm` should be ignored.
- `resp` (*mosek.rescode*) – Is the response code from the remote solver.
- `trm` (*mosek.rescode*) – Is either *rescode.ok* or a termination response code.

Groups

Remote optimization

`Task.asyncoptimize`

```
asyncoptimize(address,accesstoken) -> (token)
```

Offload the optimization task to an instance of OptServer specified by `addr`, which should be a valid URL, for example `http://server:port` or `https://server:port`. The call will exit immediately.

If the server requires authentication, the authentication token can be passed in the `accesstoken` argument.

If the server requires encryption, the keys can be passed using one of the solver parameters *sparam.remote_tls_cert* or *sparam.remote_tls_cert_path*.

The function returns a token which should be used in future calls to identify the task.

Parameters

- `address` (`str`) – Address of the OptServer. (input)
- `accesstoken` (`str`) – Access token. (input)

Return

`token` (`str`) – Returns the task token.

Groups

Remote optimization

`Task.asyncpoll`

```
asyncpoll(address,accesstoken,token) -> (respavailable,resp,trm)
```

Requests information about the status of the remote job identified by the argument `token`. For other arguments see *Task.asyncoptimize*.

Parameters

- `address` (`str`) – Address of the OptServer. (input)
- `accesstoken` (`str`) – Access token. (input)
- `token` (`str`) – The task token. (input)

Return

- `respavailable` (`bool`) – Indicates if a remote response is available. If this is not true, `resp` and `trm` should be ignored.
- `resp` (*`mosek.rescode`*) – Is the response code from the remote solver.
- `trm` (*`mosek.rescode`*) – Is either *`rescode.ok`* or a termination response code.

Groups

Remote optimization

`Task.asyncstop`

```
asyncstop(address,accesstoken,token)
```

Request that the remote job identified by `token` is terminated. For other arguments see *`Task.asyncoptimize`*.

Parameters

- `address` (`str`) – Address of the OptServer. (input)
- `accesstoken` (`str`) – Access token. (input)
- `token` (`str`) – The task token. (input)

Groups

Remote optimization

`Task.basiscond`

```
basiscond() -> (nrmbasis,nrminvbasis)
```

If a basic solution is available and it defines a nonsingular basis, then this function computes the 1-norm estimate of the basis matrix and a 1-norm estimate for the inverse of the basis matrix. The 1-norm estimates are computed using the method outlined in [Ste98], pp. 388-391.

By definition the 1-norm condition number of a matrix B is defined as

$$\kappa_1(B) := \|B\|_1 \|B^{-1}\|_1.$$

Moreover, the larger the condition number is the harder it is to solve linear equation systems involving B. Given estimates for $\|B\|_1$ and $\|B^{-1}\|_1$ it is also possible to estimate $\kappa_1(B)$.

Return

- `nrmbasis` (`float`) – An estimate for the 1-norm of the basis.
- `nrminvbasis` (`float`) – An estimate for the 1-norm of the inverse of the basis.

Groups

Solving systems with basis matrix

`Task.checkmem`

```
checkmem(file,line)
```

Checks the memory allocated by the task.

Parameters

- `file` (`str`) – File from which the function is called. (input)
- `line` (`int32`) – Line in the file from which the function is called. (input)

Groups

System, memory and debugging

`Task.chgconbound`

```
chgconbound(i,lower,finite,value)
```

Changes a bound for one constraint.

If `lower` is non-zero, then the lower bound is changed as follows:

$$\text{new lower bound} = \begin{cases} -\infty, & \texttt{finite} = 0, \\ \texttt{value} & \text{otherwise.} \end{cases}$$

Otherwise if `lower` is zero, then

$$\text{new upper bound} = \begin{cases} \infty, & \texttt{finite} = 0, \\ \texttt{value} & \text{otherwise.} \end{cases}$$

Please note that this function automatically updates the bound key for the bound, in particular, if the lower and upper bounds are identical, the bound key is changed to `fixed`.

Parameters

- `i` (`int32`) – Index of the constraint for which the bounds should be changed. (input)
- `lower` (`int32`) – If non-zero, then the lower bound is changed, otherwise the upper bound is changed. (input)
- `finite` (`int32`) – If non-zero, then `value` is assumed to be finite. (input)
- `value` (`float`) – New value for the bound. (input)

Groups

Problem data - bounds, *Problem data - constraints*, *Problem data - linear part*

`Task.chgvarbound`

```
chgvarbound(j,lower,finite,value)
```

Changes a bound for one variable.

If `lower` is non-zero, then the lower bound is changed as follows:

$$\text{new lower bound} = \begin{cases} -\infty, & \texttt{finite} = 0, \\ \texttt{value} & \text{otherwise.} \end{cases}$$

Otherwise if `lower` is zero, then

$$\text{new upper bound} = \begin{cases} \infty, & \texttt{finite} = 0, \\ \texttt{value} & \text{otherwise.} \end{cases}$$

Please note that this function automatically updates the bound key for the bound, in particular, if the lower and upper bounds are identical, the bound key is changed to `fixed`.

Parameters

- `j` (`int32`) – Index of the variable for which the bounds should be changed. (input)
- `lower` (`int32`) – If non-zero, then the lower bound is changed, otherwise the upper bound is changed. (input)
- `finite` (`int32`) – If non-zero, then `value` is assumed to be finite. (input)
- `value` (`float`) – New value for the bound. (input)

Groups

Problem data - bounds, *Problem data - variables*, *Problem data - linear part*

`Task.commitchanges`

```
commitchanges()
```

Commits all cached problem changes to the task. It is usually not necessary to call this function explicitly since changes will be committed automatically when required.

Groups

Environment and task management

`Task.deletesolution`

```
deletesolution(whichsol)
```

Undefine a solution and free the memory it uses.

Parameters

`whichsol` (*mosek.soltype*) – Selects a solution. (input)

Groups

Environment and task management, *Solution information*

`Task.dualsensitivity`

```
dualsensitivity(subj,
                leftpricej,
                rightpricej,
                leftrangej,
                rightrangej)
```

```
dualsensitivity(subj) ->
               (leftpricej,
                rightpricej,
                leftrangej,
                rightrangej)
```

Calculates sensitivity information for objective coefficients. The indexes of the coefficients to analyze are

$$\{\texttt{subj}[i] \mid i = 0, \ldots, \texttt{numj} - 1\}$$

The type of sensitivity analysis to perform (basis or optimal partition) is controlled by the parameter *iparam.sensitivity_type*.

For an example, please see Section *Example: Sensitivity Analysis*.

Parameters

- `subj` (`int32[]`) – Indexes of objective coefficients to analyze. (input)
- `leftpricej` (`float[]`) – `leftpricej`$[j]$ is the left shadow price for the coefficient with index `subj`[j]. (output)
- `rightpricej` (`float[]`) – `rightpricej`$[j]$ is the right shadow price for the coefficient with index `subj`[j]. (output)
- `leftrangej` (`float[]`) – `leftrangej`$[j]$ is the left range β_1 for the coefficient with index `subj`[j]. (output)
- `rightrangej` (`float[]`) – `rightrangej`$[j]$ is the right range β_2 for the coefficient with index `subj`[j]. (output)

Return

- `leftpricej` (`float[]`) – `leftpricej`$[j]$ is the left shadow price for the coefficient with index `subj`[j].
- `rightpricej` (`float[]`) – `rightpricej`$[j]$ is the right shadow price for the coefficient with index `subj`[j].
- `leftrangej` (`float[]`) – `leftrangej`$[j]$ is the left range β_1 for the coefficient with index `subj`[j].
- `rightrangej` (`float[]`) – `rightrangej`$[j]$ is the right range β_2 for the coefficient with index `subj`[j].

Groups

Sensitivity analysis

`Task.emptyafebarfrow`

```
emptyafebarfrow(afeidx)
```

Clears a row in $\overline{F}$ i.e. sets $\overline{F}_{\text{afeidx},*} = 0$.

Parameters
`afeidx` (`int64`) – Row index of $\overline{F}$. (input)

Groups
Problem data - affine expressions, *Problem data - semidefinite*

`Task.emptyafebarfrowlist`

```
emptyafebarfrowlist(afeidxlist)
```

Clears a number of rows in $\overline{F}$ i.e. sets $\overline{F}_{i,*} = 0$ for all indices i in `afeidxlist`.

Parameters
`afeidxlist` (`int64[]`) – Indices of rows in $\overline{F}$ to clear. (input)

Groups
Problem data - affine expressions, *Problem data - semidefinite*

`Task.emptyafefcol`

```
emptyafefcol(varidx)
```

Clears one column in the affine constraint matrix F, that is sets $F_{*,\text{varidx}} = 0$.

Parameters
`varidx` (`int32`) – Index of a variable (column in F). (input)

Groups
Problem data - affine expressions

`Task.emptyafefcollist`

```
emptyafefcollist(varidx)
```

Clears a number of columns in F i.e. sets $F_{*,j} = 0$ for all indices j in `varidx`.

Parameters
`varidx` (`int32[]`) – Indices of variables (columns) in F to clear. (input)

Groups
Problem data - affine expressions

`Task.emptyafefrow`

```
emptyafefrow(afeidx)
```

Clears one row in the affine constraint matrix F, that is sets $F_{\text{afeidx},*} = 0$.

Parameters
`afeidx` (`int64`) – Index of a row in F. (input)

Groups
Problem data - affine expressions

`Task.emptyafefrowlist`

```
emptyafefrowlist(afeidx)
```

Clears a number of rows in F i.e. sets $F_{i,*} = 0$ for all indices i in `afeidx`.

Parameters
`afeidx` (`int64[]`) – Indices of rows in F to clear. (input)

Groups
Problem data - affine expressions

`Task.evaluateacc`

```
evaluateacc(whichsol,accidx,activity)
```

```
evaluateacc(whichsol,accidx) -> (activity)
```

Evaluates the activity of an affine conic constraint.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `accidx` (`int64`) – The index of the affine conic constraint. (input)
- `activity` (`float[]`) – The activity of the affine conic constraint. The array should have length equal to the dimension of the constraint. (output)

Return

`activity` (`float[]`) – The activity of the affine conic constraint. The array should have length equal to the dimension of the constraint.

Groups

Solution - primal, *Problem data - affine conic constraints*

`Task.evaluateaccs`

```
evaluateaccs(whichsol,activity)
```

```
evaluateaccs(whichsol) -> (activity)
```

Evaluates the activities of all affine conic constraints.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `activity` (`float[]`) – The activity of affine conic constraints. The array should have length equal to the sum of dimensions of all affine conic constraints. (output)

Return

`activity` (`float[]`) – The activity of affine conic constraints. The array should have length equal to the sum of dimensions of all affine conic constraints.

Groups

Solution - primal, *Problem data - affine conic constraints*

`Task.generateaccnames`

```
generateaccnames(sub,fmt,dims,sp,namedaxisidxs,names)
```

Internal.

Parameters

- `sub` (`int64[]`) – Indexes of the affine conic constraints. (input)
- `fmt` (`str`) – The variable name formatting string. (input)
- `dims` (`int32[]`) – Dimensions in the shape. (input)
- `sp` (`int64[]`) – Items that should be named. (input)
- `namedaxisidxs` (`int32[]`) – List if named index axes (input)
- `names` (`str[]`) – All axis names. (input)

Groups

Names

`Task.generatebarvarnames`

```
generatebarvarnames(subj,fmt,dims,sp,namedaxisidxs,names)
```

Internal.

Parameters

- `subj` (`int32[]`) – Indexes of the variables. (input)
- `fmt` (`str`) – The variable name formatting string. (input)
- `dims` (`int32[]`) – Dimensions in the shape. (input)
- `sp` (`int64[]`) – Items that should be named. (input)
- `namedaxisidxs` (`int32[]`) – List if named index axes (input)
- `names` (`str[]`) – All axis names. (input)

Groups

Names, *Problem data - variables*, *Problem data - linear part*

~~`Task.generateconenames`~~ *Deprecated*

```
generateconenames(subk,fmt,dims,sp,namedaxisidxs,names)
```

Internal, deprecated.

Parameters

- `subk` (`int32[]`) – Indexes of the cone. (input)
- `fmt` (`str`) – The cone name formatting string. (input)
- `dims` (`int32[]`) – Dimensions in the shape. (input)
- `sp` (`int64[]`) – Items that should be named. (input)
- `namedaxisidxs` (`int32[]`) – List if named index axes (input)
- `names` (`str[]`) – All axis names. (input)

Groups

Names, *Problem data - cones (deprecated)*

`Task.generateconnames`

```
generateconnames(subi,fmt,dims,sp,namedaxisidxs,names)
```

Internal.

Parameters

- `subi` (`int32[]`) – Indexes of the constraints. (input)
- `fmt` (`str`) – The constraint name formatting string. (input)
- `dims` (`int32[]`) – Dimensions in the shape. (input)
- `sp` (`int64[]`) – Items that should be named. (input)
- `namedaxisidxs` (`int32[]`) – List if named index axes (input)
- `names` (`str[]`) – All axis names. (input)

Groups

Names, *Problem data - constraints*, *Problem data - linear part*

`Task.generatedjcnames`

```
generatedjcnames(sub,fmt,dims,sp,namedaxisidxs,names)
```

Internal.

Parameters

- `sub` (`int64[]`) – Indexes of the disjunctive constraints. (input)
- `fmt` (`str`) – The variable name formatting string. (input)
- `dims` (`int32[]`) – Dimensions in the shape. (input)
- `sp` (`int64[]`) – Items that should be named. (input)

- `namedaxisidxs` (`int32[]`) – List if named index axes (input)
- `names` (`str[]`) – All axis names. (input)

Groups

Names

`Task.generatevarnames`

```
generatevarnames(subj,fmt,dims,sp,namedaxisidxs,names)
```

Internal.

Parameters

- `subj` (`int32[]`) – Indexes of the variables. (input)
- `fmt` (`str`) – The variable name formatting string. (input)
- `dims` (`int32[]`) – Dimensions in the shape. (input)
- `sp` (`int64[]`) – Items that should be named. (input)
- `namedaxisidxs` (`int32[]`) – List if named index axes (input)
- `names` (`str[]`) – All axis names. (input)

Groups

Names, *Problem data - variables*, *Problem data - linear part*

`Task.getaccafeidxlist`

```
getaccafeidxlist(accidx,afeidxlist)
```

```
getaccafeidxlist(accidx) -> (afeidxlist)
```

Obtains the list of affine expressions appearing in the affine conic constraint.

Parameters

- `accidx` (`int64`) – Index of the affine conic constraint. (input)
- `afeidxlist` (`int64[]`) – List of indexes of affine expressions appearing in the constraint. (output)

Return

`afeidxlist` (`int64[]`) – List of indexes of affine expressions appearing in the constraint.

Groups

Problem data - affine conic constraints, *Inspecting the task*

`Task.getaccb`

```
getaccb(accidx,b)
```

```
getaccb(accidx) -> (b)
```

Obtains the additional constant term vector appearing in the affine conic constraint.

Parameters

- `accidx` (`int64`) – Index of the affine conic constraint. (input)
- `b` (`float[]`) – The vector b appearing in the constraint. (output)

Return

`b` (`float[]`) – The vector b appearing in the constraint.

Groups

Problem data - affine conic constraints, *Inspecting the task*

`Task.getaccbarfblocktriplet`

```
getaccbarfblocktriplet(acc_afe,bar_var,blk_row,blk_col,blk_val) -> (numtrip)
```

```
getaccbarfblocktriplet() ->
                       (numtrip,
                        acc_afe,
                        bar_var,
                        blk_row,
                        blk_col,
                        blk_val)
```

Obtains $\overline{F}$, implied by the ACCs, in block triplet form. If the AFEs passed to the ACCs were out of order, then this function can be used to obtain the barF as seen by the ACCs.

Parameters

- `acc_afe` (`int64[]`) – Index of the AFE within the concatenated list of AFEs in ACCs. (output)
- `bar_var` (`int32[]`) – Symmetric matrix variable index. (output)
- `blk_row` (`int32[]`) – Block row index. (output)
- `blk_col` (`int32[]`) – Block column index. (output)
- `blk_val` (`float[]`) – The numerical value associated with each block triplet. (output)

Return

- `numtrip` (`int64`) – Number of elements in the block triplet form.
- `acc_afe` (`int64[]`) – Index of the AFE within the concatenated list of AFEs in ACCs.
- `bar_var` (`int32[]`) – Symmetric matrix variable index.
- `blk_row` (`int32[]`) – Block row index.
- `blk_col` (`int32[]`) – Block column index.
- `blk_val` (`float[]`) – The numerical value associated with each block triplet.

Groups

Problem data - affine expressions, *Problem data - semidefinite*

`Task.getaccbarfnumblocktriplets`

```
getaccbarfnumblocktriplets() -> (numtrip)
```

Obtains an upper bound on the number of elements in the block triplet form of $\overline{F}$, as used within the ACCs.

Return

`numtrip` (`int64`) – An upper bound on the number of elements in the block triplet form of $\overline{F}$., as used within the ACCs.

Groups

Problem data - semidefinite, *Problem data - affine conic constraints*, *Inspecting the task*

`Task.getaccdomain`

```
getaccdomain(accidx) -> (domidx)
```

Obtains the domain appearing in the affine conic constraint.

Parameters

`accidx` (`int64`) – The index of the affine conic constraint. (input)

Return

`domidx` (`int64`) – The index of domain in the affine conic constraint.

Groups

Problem data - affine conic constraints, *Inspecting the task*

`Task.getaccdoty`

```
getaccdoty(whichsol,accidx,doty)
```

```
getaccdoty(whichsol,accidx) -> (doty)
```

Obtains the $\dot{y}$ vector for a solution (the dual values of an affine conic constraint).

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `accidx` (`int64`) – The index of the affine conic constraint. (input)
- `doty` (`float[]`) – The dual values for this affine conic constraint. The array should have length equal to the dimension of the constraint. (output)

Return

`doty` (`float[]`) – The dual values for this affine conic constraint. The array should have length equal to the dimension of the constraint.

Groups

Solution - dual, *Problem data - affine conic constraints*

`Task.getaccdotys`

```
getaccdotys(whichsol,doty)
```

```
getaccdotys(whichsol) -> (doty)
```

Obtains the $\dot{y}$ vector for a solution (the dual values of all affine conic constraint).

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `doty` (`float[]`) – The dual values of affine conic constraints. The array should have length equal to the sum of dimensions of all affine conic constraints. (output)

Return

`doty` (`float[]`) – The dual values of affine conic constraints. The array should have length equal to the sum of dimensions of all affine conic constraints.

Groups

Solution - dual, *Problem data - affine conic constraints*

`Task.getaccfnumnz`

```
getaccfnumnz() -> (accfnnz)
```

If the AFEs are not added sequentially to the ACCs, then the present function gives the number of nonzero elements in the F matrix that would be implied by the ordering of AFEs within ACCs.

Return

`accfnnz` (`int64`) – Number of non-zeros in F implied by ACCs.

Groups

Problem data - affine conic constraints, *Inspecting the task*

`Task.getaccftrip`

```
getaccftrip(frow,fcol,fval)
```

```
getaccftrip() -> (frow,fcol,fval)
```

Obtains the F (that would be implied by the ordering of the AFEs within the ACCs) in triplet format.

Parameters

- `frow` (`int64[]`) – Row indices of nonzeros in the implied F matrix. (output)
- `fcol` (`int32[]`) – Column indices of nonzeros in the implied F matrix. (output)
- `fval` (`float[]`) – Values of nonzero entries in the implied F matrix. (output)

Return

- `frow` (`int64[]`) – Row indices of nonzeros in the implied F matrix.
- `fcol` (`int32[]`) – Column indices of nonzeros in the implied F matrix.
- `fval` (`float[]`) – Values of nonzero entries in the implied F matrix.

Groups

Problem data - affine conic constraints, *Inspecting the task*

`Task.getaccgvector`

```
getaccgvector(g)
```

```
getaccgvector() -> (g)
```

If the AFEs are passed out of sequence to the ACCs, then this function can be used to obtain the vector g of constant terms used within the ACCs.

Parameters

`g` (`float[]`) – The g used within the ACCs as a dense vector. The length is sum of the dimensions of the ACCs. (output)

Return

`g` (`float[]`) – The g used within the ACCs as a dense vector. The length is sum of the dimensions of the ACCs.

Groups

Inspecting the task, *Problem data - affine conic constraints*

`Task.getaccn`

```
getaccn(accidx) -> (n)
```

Obtains the dimension of the affine conic constraint.

Parameters

`accidx` (`int64`) – The index of the affine conic constraint. (input)

Return

`n` (`int64`) – The dimension of the affine conic constraint (equal to the dimension of its domain).

Groups

Problem data - affine conic constraints, *Inspecting the task*

`Task.getaccname`

```
getaccname(accidx) -> (name)
```

Obtains the name of an affine conic constraint.

Parameters

`accidx` (`int64`) – Index of an affine conic constraint. (input)

Return

`name` (`str`) – Returns the required name.

Groups

Names, *Problem data - affine conic constraints*, *Inspecting the task*

`Task.getaccnamelen`

```
getaccnamelen(accidx) -> (len)
```

Obtains the length of the name of an affine conic constraint.

Parameters
`accidx` (`int64`) – Index of an affine conic constraint. (input)

Return
`len` (`int32`) – Returns the length of the indicated name.

Groups
Names, *Problem data - affine conic constraints*, *Inspecting the task*

`Task.getaccntot`

```
getaccntot() -> (n)
```

Obtains the total dimension of all affine conic constraints (the sum of all their dimensions).

Return
`n` (`int64`) – The total dimension of all affine conic constraints.

Groups
Problem data - affine conic constraints, *Inspecting the task*

`Task.getaccs`

```
getaccs(domidxlist,afeidxlist,b)
```

```
getaccs() -> (domidxlist,afeidxlist,b)
```

Obtains full data of all affine conic constraints. The output array `domainidxlist` must have at least length determined by *`Task.getnumacc`*. The output arrays `afeidxlist` and `b` must have at least length determined by *`Task.getaccntot`*.

Parameters
- `domidxlist` (`int64[]`) – The list of domains appearing in all affine conic constraints. (output)
- `afeidxlist` (`int64[]`) – The concatenation of index lists of affine expressions appearing in all affine conic constraints. (output)
- `b` (`float[]`) – The concatenation of vectors b appearing in all affine conic constraints. (output)

Return
- `domidxlist` (`int64[]`) – The list of domains appearing in all affine conic constraints.
- `afeidxlist` (`int64[]`) – The concatenation of index lists of affine expressions appearing in all affine conic constraints.
- `b` (`float[]`) – The concatenation of vectors b appearing in all affine conic constraints.

Groups
Problem data - affine conic constraints, *Inspecting the task*

`Task.getacol`

```
getacol(j,subj,valj) -> (nzj)
```

```
getacol(j) -> (nzj,subj,valj)
```

Obtains one column of A in a sparse format.

Parameters

- `j` (`int32`) – Index of the column. (input)
- `subj` (`int32[]`) – Row indices of the non-zeros in the column obtained. (output)
- `valj` (`float[]`) – Numerical values in the column obtained. (output)

Return

- `nzj` (`int32`) – Number of non-zeros in the column obtained.
- `subj` (`int32[]`) – Row indices of the non-zeros in the column obtained.
- `valj` (`float[]`) – Numerical values in the column obtained.

Groups

Problem data - linear part, *Inspecting the task*

`Task.getacolnumnz`

```
getacolnumnz(i) -> (nzj)
```

Obtains the number of non-zero elements in one column of A.

Parameters

`i` (`int32`) – Index of the column. (input)

Return

`nzj` (`int32`) – Number of non-zeros in the j-th column of A.

Groups

Problem data - linear part, *Inspecting the task*

`Task.getacolslice`

```
getacolslice(first,last,ptrb,ptre,sub,val)
```

```
getacolslice(first,last) -> (ptrb,ptre,sub,val)
```

Obtains a sequence of columns from A in sparse format.

Parameters

- `first` (`int32`) – Index of the first column in the sequence. (input)
- `last` (`int32`) – Index of the last column in the sequence **plus one**. (input)
- `ptrb` (`int64[]`) – `ptrb[t]` is an index pointing to the first element in the t-th column obtained. (output)
- `ptre` (`int64[]`) – `ptre[t]` is an index pointing to the last element plus one in the t-th column obtained. (output)
- `sub` (`int32[]`) – Contains the row subscripts. (output)
- `val` (`float[]`) – Contains the coefficient values. (output)

Return

- `ptrb` (`int64[]`) – `ptrb[t]` is an index pointing to the first element in the t-th column obtained.
- `ptre` (`int64[]`) – `ptre[t]` is an index pointing to the last element plus one in the t-th column obtained.
- `sub` (`int32[]`) – Contains the row subscripts.
- `val` (`float[]`) – Contains the coefficient values.

Groups

Problem data - linear part, *Inspecting the task*

`Task.getacolslicenumnz`

```
getacolslicenumnz(first,last) -> (numnz)
```

Obtains the number of non-zeros in a slice of columns of A.

Parameters

- `first` (`int32`) – Index of the first column in the sequence. (input)
- `last` (`int32`) – Index of the last column **plus one** in the sequence. (input)

Return

`numnz` (`int64`) – Number of non-zeros in the slice.

Groups

Problem data - linear part, *Inspecting the task*

`Task.getacolslicetrip`

```
getacolslicetrip(first,last,subi,subj,val)
```

```
getacolslicetrip(first,last) -> (subi,subj,val)
```

Obtains a sequence of columns from A in sparse triplet format. The function returns the content of all columns whose index `j` satisfies `first <= j < last`. The triplets corresponding to nonzero entries are stored in the arrays `subi`, `subj` and `val`.

Parameters

- `first` (`int32`) – Index of the first column in the sequence. (input)
- `last` (`int32`) – Index of the last column in the sequence **plus one**. (input)
- `subi` (`int32[]`) – Constraint subscripts. (output)
- `subj` (`int32[]`) – Column subscripts. (output)
- `val` (`float[]`) – Values. (output)

Return

- `subi` (`int32[]`) – Constraint subscripts.
- `subj` (`int32[]`) – Column subscripts.
- `val` (`float[]`) – Values.

Groups

Problem data - linear part, *Inspecting the task*

`Task.getafebarfblocktriplet`

```
getafebarfblocktriplet(afeidx,barvaridx,subk,subl,valkl) -> (numtrip)
```

```
getafebarfblocktriplet() ->
                      (numtrip,
                       afeidx,
                       barvaridx,
                       subk,
                       subl,
                       valkl)
```

Obtains $\overline{F}$ in block triplet form.

Parameters

- `afeidx` (`int64[]`) – Constraint index. (output)
- `barvaridx` (`int32[]`) – Symmetric matrix variable index. (output)
- `subk` (`int32[]`) – Block row index. (output)
- `subl` (`int32[]`) – Block column index. (output)
- `valkl` (`float[]`) – The numerical value associated with each block triplet. (output)

Return

- `numtrip` (`int64`) – Number of elements in the block triplet form.
- `afeidx` (`int64[]`) – Constraint index.
- `barvaridx` (`int32[]`) – Symmetric matrix variable index.
- `subk` (`int32[]`) – Block row index.

- `subl` (`int32[]`) – Block column index.
- `valkl` (`float[]`) – The numerical value associated with each block triplet.

Groups

Problem data - affine expressions, *Problem data - semidefinite*

`Task.getafebarfnumblocktriplets`

```
getafebarfnumblocktriplets() -> (numtrip)
```

Obtains an upper bound on the number of elements in the block triplet form of $\overline{F}$.

Return

`numtrip` (`int64`) – An upper bound on the number of elements in the block triplet form of $\overline{F}$.

Groups

Problem data - semidefinite, *Inspecting the task*

`Task.getafebarfnumrowentries`

```
getafebarfnumrowentries(afeidx) -> (numentr)
```

Obtains the number of nonzero entries in one row of $\overline{F}$, that is the number of j such that $\overline{F}_{\text{afeidx},j}$ is not the zero matrix.

Parameters

`afeidx` (`int64`) – Row index of $\overline{F}$. (input)

Return

`numentr` (`int32`) – Number of nonzero entries in a row of $\overline{F}$.

Groups

Problem data - affine expressions, *Problem data - semidefinite*, *Inspecting the task*

`Task.getafebarfrow`

```
getafebarfrow(afeidx,
              barvaridx,
              ptrterm,
              numterm,
              termidx,
              termweight)
```

```
getafebarfrow(afeidx) ->
             (barvaridx,
              ptrterm,
              numterm,
              termidx,
              termweight)
```

Obtains all nonzero entries in one row $\overline{F}_{\text{afeidx},*}$ of $\overline{F}$. For every k there is a nonzero entry $\overline{F}_{\text{afeidx,barvaridx}[k]}$, which is represented as a weighted sum of numterm$[k]$ terms. The indices in the matrix store E and their weights for the k-th entry appear in the arrays `termidx` and `termweight` in positions

$$\text{ptrterm}[k], \ldots, \text{ptrterm}[k] + (\text{numterm}[k] - 1).$$

The arrays should be long enough to accommodate the data; their required lengths can be obtained with *`Task.getafebarfrowinfo`*.

Parameters

- `afeidx` (`int64`) – Row index of $\overline{F}$. (input)

- `barvaridx` (`int32[]`) – Semidefinite variable indices of nonzero entries in the row of $\overline{F}$. (output)
- `ptrterm` (`int64[]`) – Pointers to the start of each entry's description. (output)
- `numterm` (`int64[]`) – Number of terms in the weighted sum representation of each entry. (output)
- `termidx` (`int64[]`) – Indices of semidefinite matrices from the matrix store E. (output)
- `termweight` (`float[]`) – Weights appearing in the weighted sum representations of all entries. (output)

Return

- `barvaridx` (`int32[]`) – Semidefinite variable indices of nonzero entries in the row of $\overline{F}$.
- `ptrterm` (`int64[]`) – Pointers to the start of each entry's description.
- `numterm` (`int64[]`) – Number of terms in the weighted sum representation of each entry.
- `termidx` (`int64[]`) – Indices of semidefinite matrices from the matrix store E.
- `termweight` (`float[]`) – Weights appearing in the weighted sum representations of all entries.

Groups

Problem data - affine expressions, *Problem data - semidefinite*, *Inspecting the task*

`Task.getafebarfrowinfo`

```
getafebarfrowinfo(afeidx) -> (numentr,numterm)
```

Obtains information about one row of $\overline{F}$: the number of nonzero entries, that is the number of j such that $\overline{F}_{\text{afeidx},j}$ is not the zero matrix, as well as the total number of terms in the representations of all these entries as weighted sums of matrices from E. This information provides the data sizes required for a call to *`Task.getafebarfrow`*.

Parameters

`afeidx` (`int64`) – Row index of $\overline{F}$. (input)

Return

- `numentr` (`int32`) – Number of nonzero entries in a row of $\overline{F}$.
- `numterm` (`int64`) – Number of terms in the weighted sums representation of the row of $\overline{F}$.

Groups

Problem data - affine expressions, *Problem data - semidefinite*, *Inspecting the task*

`Task.getafefnumnz`

```
getafefnumnz() -> (numnz)
```

Obtains the total number of nonzeros in F.

Return

`numnz` (`int64`) – Number of non-zeros in F.

Groups

Problem data - affine expressions, *Inspecting the task*

`Task.getafefrow`

```
getafefrow(afeidx,varidx,val) -> (numnz)
```

```
getafefrow(afeidx) -> (numnz,varidx,val)
```

Obtains one row of F in sparse format.

Parameters

- `afeidx` (`int64`) – Index of a row in F. (input)
- `varidx` (`int32[]`) – Column indices of the non-zeros in the row obtained. (output)
- `val` (`float[]`) – Values of the non-zeros in the row obtained. (output)

Return

- `numnz` (`int32`) – Number of non-zeros in the row obtained.
- `varidx` (`int32[]`) – Column indices of the non-zeros in the row obtained.
- `val` (`float[]`) – Values of the non-zeros in the row obtained.

Groups

Problem data - affine expressions, *Inspecting the task*

`Task.getafefrownumnz`

```
getafefrownumnz(afeidx) -> (numnz)
```

Obtains the number of nonzeros in one row of F.

Parameters

`afeidx` (`int64`) – Index of a row in F. (input)

Return

`numnz` (`int32`) – Number of non-zeros in row `afeidx` of F.

Groups

Problem data - affine expressions, *Inspecting the task*

`Task.getafeftrip`

```
getafeftrip(afeidx,varidx,val)
```

```
getafeftrip() -> (afeidx,varidx,val)
```

Obtains the F in triplet format.

Parameters

- `afeidx` (`int64[]`) – Row indices of nonzeros. (output)
- `varidx` (`int32[]`) – Column indices of nonzeros. (output)
- `val` (`float[]`) – Values of nonzero entries. (output)

Return

- `afeidx` (`int64[]`) – Row indices of nonzeros.
- `varidx` (`int32[]`) – Column indices of nonzeros.
- `val` (`float[]`) – Values of nonzero entries.

Groups

Problem data - affine expressions, *Inspecting the task*

`Task.getafeg`

```
getafeg(afeidx) -> (g)
```

Obtains a single coefficient in g.

Parameters

`afeidx` (`int64`) – Index of an element in g. (input)

Return

`g` (`float`) – The value of g_{afeidx}.

Groups

Problem data - affine expressions, *Inspecting the task*

`Task.getafegslice`

```
getafegslice(first,last,g)
```

```
getafegslice(first,last) -> (g)
```

Obtains a sequence of elements from the vector g of constant terms in the affine expressions list.

Parameters

- `first` (`int64`) – First index in the sequence. (input)
- `last` (`int64`) – Last index plus 1 in the sequence. (input)
- `g` (`float[]`) – The slice g as a dense vector. The length is `last-first`. (output)

Return

`g` (`float[]`) – The slice g as a dense vector. The length is `last-first`.

Groups

Inspecting the task, *Problem data - affine expressions*

`Task.getaij`

```
getaij(i,j) -> (aij)
```

Obtains a single coefficient in A.

Parameters

- `i` (`int32`) – Row index of the coefficient to be returned. (input)
- `j` (`int32`) – Column index of the coefficient to be returned. (input)

Return

`aij` (`float`) – The required coefficient $a_{i,j}$.

Groups

Problem data - linear part, *Inspecting the task*

`Task.getapiecenumnz`

```
getapiecenumnz(firsti,lasti,firstj,lastj) -> (numnz)
```

Obtains the number non-zeros in a rectangular piece of A, i.e. the number of elements in the set

$$\{(i,j) \ : \ a_{i,j} \neq 0, \ \texttt{firsti} \leq i \leq \texttt{lasti} - 1, \ \texttt{firstj} \leq j \leq \texttt{lastj} - 1\}$$

This function is not an efficient way to obtain the number of non-zeros in one row or column. In that case use the function *`Task.getarownumnz`* or *`Task.getacolnumnz`*.

Parameters

- `firsti` (`int32`) – Index of the first row in the rectangular piece. (input)
- `lasti` (`int32`) – Index of the last row plus one in the rectangular piece. (input)
- `firstj` (`int32`) – Index of the first column in the rectangular piece. (input)
- `lastj` (`int32`) – Index of the last column plus one in the rectangular piece. (input)

Return

`numnz` (`int32`) – Number of non-zero A elements in the rectangular piece.

Groups

Problem data - linear part, *Inspecting the task*

`Task.getarow`

```
getarow(i,subi,vali) -> (nzi)
```

```
getarow(i) -> (nzi,subi,vali)
```

Obtains one row of A in a sparse format.

Parameters

- `i` (`int32`) – Index of the row. (input)
- `subi` (`int32[]`) – Column indices of the non-zeros in the row obtained. (output)
- `vali` (`float[]`) – Numerical values of the row obtained. (output)

Return

- `nzi` (`int32`) – Number of non-zeros in the row obtained.
- `subi` (`int32[]`) – Column indices of the non-zeros in the row obtained.
- `vali` (`float[]`) – Numerical values of the row obtained.

Groups

Problem data - linear part, *Inspecting the task*

`Task.getarownumnz`

```
getarownumnz(i) -> (nzi)
```

Obtains the number of non-zero elements in one row of A.

Parameters

`i` (`int32`) – Index of the row. (input)

Return

`nzi` (`int32`) – Number of non-zeros in the i-th row of A.

Groups

Problem data - linear part, *Inspecting the task*

`Task.getarowslice`

```
getarowslice(first,last,ptrb,ptre,sub,val)
```

```
getarowslice(first,last) -> (ptrb,ptre,sub,val)
```

Obtains a sequence of rows from A in sparse format.

Parameters

- `first` (`int32`) – Index of the first row in the sequence. (input)
- `last` (`int32`) – Index of the last row in the sequence **plus one**. (input)
- `ptrb` (`int64[]`) – `ptrb[t]` is an index pointing to the first element in the t-th row obtained. (output)
- `ptre` (`int64[]`) – `ptre[t]` is an index pointing to the last element plus one in the t-th row obtained. (output)
- `sub` (`int32[]`) – Contains the column subscripts. (output)
- `val` (`float[]`) – Contains the coefficient values. (output)

Return

- `ptrb` (`int64[]`) – `ptrb[t]` is an index pointing to the first element in the t-th row obtained.
- `ptre` (`int64[]`) – `ptre[t]` is an index pointing to the last element plus one in the t-th row obtained.
- `sub` (`int32[]`) – Contains the column subscripts.
- `val` (`float[]`) – Contains the coefficient values.

Groups

Problem data - linear part, *Inspecting the task*

`Task.getarowslicenumnz`

```
getarowslicenumnz(first,last) -> (numnz)
```

Obtains the number of non-zeros in a slice of rows of A.

Parameters

- `first` (`int32`) – Index of the first row in the sequence. (input)
- `last` (`int32`) – Index of the last row **plus one** in the sequence. (input)

Return

`numnz` (`int64`) – Number of non-zeros in the slice.

Groups

Problem data - linear part, *Inspecting the task*

`Task.getarowslicetrip`

```
getarowslicetrip(first,last,subi,subj,val)
```

```
getarowslicetrip(first,last) -> (subi,subj,val)
```

Obtains a sequence of rows from A in sparse triplet format. The function returns the content of all rows whose index `i` satisfies `first <= i < last`. The triplets corresponding to nonzero entries are stored in the arrays `subi`, `subj` and `val`.

Parameters

- `first` (`int32`) – Index of the first row in the sequence. (input)
- `last` (`int32`) – Index of the last row in the sequence **plus one**. (input)
- `subi` (`int32[]`) – Constraint subscripts. (output)
- `subj` (`int32[]`) – Column subscripts. (output)
- `val` (`float[]`) – Values. (output)

Return

- `subi` (`int32[]`) – Constraint subscripts.
- `subj` (`int32[]`) – Column subscripts.
- `val` (`float[]`) – Values.

Groups

Problem data - linear part, *Inspecting the task*

`Task.getatrip`

```
getatrip(subi,subj,val)
```

```
getatrip() -> (subi,subj,val)
```

Obtains A in sparse triplet format. The triplets corresponding to nonzero entries are stored in the arrays `subi`, `subj` and `val`.

Parameters

- `subi` (`int32[]`) – Constraint subscripts. (output)
- `subj` (`int32[]`) – Column subscripts. (output)
- `val` (`float[]`) – Values. (output)

Return

- `subi` (`int32[]`) – Constraint subscripts.
- `subj` (`int32[]`) – Column subscripts.
- `val` (`float[]`) – Values.

Groups

Problem data - linear part, *Inspecting the task*

`Task.getatruncatetol`

```
getatruncatetol(tolzero)
```

```
getatruncatetol() -> (tolzero)
```

Obtains the tolerance value set with *Task.putatruncatetol*.

Parameters

`tolzero` (`float[]`) – All elements $|a_{i,j}|$ less than this tolerance is truncated to zero. (output)

Return

`tolzero` (`float[]`) – All elements $|a_{i,j}|$ less than this tolerance is truncated to zero.

Groups

Parameters, *Problem data - linear part*

`Task.getbarablocktriplet`

```
getbarablocktriplet(subi,subj,subk,subl,valijkl) -> (num)
```

```
getbarablocktriplet() -> (num,subi,subj,subk,subl,valijkl)
```

Obtains $\overline{A}$ in block triplet form.

Parameters

- `subi` (`int32[]`) – Constraint index. (output)
- `subj` (`int32[]`) – Symmetric matrix variable index. (output)
- `subk` (`int32[]`) – Block row index. (output)
- `subl` (`int32[]`) – Block column index. (output)
- `valijkl` (`float[]`) – The numerical value associated with each block triplet. (output)

Return

- `num` (`int64`) – Number of elements in the block triplet form.
- `subi` (`int32[]`) – Constraint index.
- `subj` (`int32[]`) – Symmetric matrix variable index.
- `subk` (`int32[]`) – Block row index.
- `subl` (`int32[]`) – Block column index.
- `valijkl` (`float[]`) – The numerical value associated with each block triplet.

Groups

Problem data - semidefinite, *Inspecting the task*

`Task.getbaraidx`

```
getbaraidx(idx,sub,weights) -> (i,j,num)
```

```
getbaraidx(idx) -> (i,j,num,sub,weights)
```

Obtains information about an element in $\overline{A}$. Since $\overline{A}$ is a sparse matrix of symmetric matrices, only the nonzero elements in $\overline{A}$ are stored in order to save space. Now $\overline{A}$ is stored vectorized i.e. as one long vector. This function makes it possible to obtain information such as the row index and the column index of a particular element of the vectorized form of $\overline{A}$.

Please observe if one element of $\overline{A}$ is inputted multiple times then it may be stored several times in vectorized form. In that case the element with the highest index is the one that is used.

Parameters

- `idx` (`int64`) – Position of the element in the vectorized form. (input)
- `sub` (`int64[]`) – A list indexes of the elements from symmetric matrix storage that appear in the weighted sum. (output)

- `weights` (`float[]`) – The weights associated with each term in the weighted sum. (output)

Return

- `i` (`int32`) – Row index of the element at position `idx`.
- `j` (`int32`) – Column index of the element at position `idx`.
- `num` (`int64`) – Number of terms in weighted sum that forms the element.
- `sub` (`int64[]`) – A list indexes of the elements from symmetric matrix storage that appear in the weighted sum.
- `weights` (`float[]`) – The weights associated with each term in the weighted sum.

Groups

Problem data - semidefinite, *Inspecting the task*

`Task.getbaraidxij`

```
getbaraidxij(idx) -> (i,j)
```

Obtains information about an element in $\overline{A}$. Since $\overline{A}$ is a sparse matrix of symmetric matrices, only the nonzero elements in $\overline{A}$ are stored in order to save space. Now $\overline{A}$ is stored vectorized i.e. as one long vector. This function makes it possible to obtain information such as the row index and the column index of a particular element of the vectorized form of $\overline{A}$.

Please note that if one element of $\overline{A}$ is inputted multiple times then it may be stored several times in vectorized form. In that case the element with the highest index is the one that is used.

Parameters

`idx` (`int64`) – Position of the element in the vectorized form. (input)

Return

- `i` (`int32`) – Row index of the element at position `idx`.
- `j` (`int32`) – Column index of the element at position `idx`.

Groups

Problem data - semidefinite, *Inspecting the task*

`Task.getbaraidxinfo`

```
getbaraidxinfo(idx) -> (num)
```

Each nonzero element in $\overline{A}_{ij}$ is formed as a weighted sum of symmetric matrices. Using this function the number of terms in the weighted sum can be obtained. See description of *Task.appendsparsesymmat* for details about the weighted sum.

Parameters

`idx` (`int64`) – The internal position of the element for which information should be obtained. (input)

Return

`num` (`int64`) – Number of terms in the weighted sum that form the specified element in $\overline{A}$.

Groups

Problem data - semidefinite, *Inspecting the task*

`Task.getbarasparsity`

```
getbarasparsity(idxij) -> (numnz)
```

```
getbarasparsity() -> (numnz,idxij)
```

The matrix $\overline{A}$ is assumed to be a sparse matrix of symmetric matrices. This implies that many of the elements in $\overline{A}$ are likely to be zero matrices. Therefore, in order to save space, only nonzero elements in $\overline{A}$ are stored on vectorized form. This function is used to obtain the sparsity pattern of $\overline{A}$ and the position of each nonzero element in the vectorized form of $\overline{A}$. From the index detailed information about each nonzero $\overline{A}_{i,j}$ can be obtained using *Task.getbaraidxinfo* and *Task.getbaraidx*.

Parameters

`idxij` (`int64[]`) – Position of each nonzero element in the vectorized form of $\overline{A}$. (output)

Return

- `numnz` (`int64`) – Number of nonzero elements in $\overline{A}$.
- `idxij` (`int64[]`) – Position of each nonzero element in the vectorized form of $\overline{A}$.

Groups

Problem data - semidefinite, *Inspecting the task*

`Task.getbarcblocktriplet`

```
getbarcblocktriplet(subj,subk,subl,valjkl) -> (num)
```

```
getbarcblocktriplet() -> (num,subj,subk,subl,valjkl)
```

Obtains $\overline{C}$ in block triplet form.

Parameters

- `subj` (`int32[]`) – Symmetric matrix variable index. (output)
- `subk` (`int32[]`) – Block row index. (output)
- `subl` (`int32[]`) – Block column index. (output)
- `valjkl` (`float[]`) – The numerical value associated with each block triplet. (output)

Return

- `num` (`int64`) – Number of elements in the block triplet form.
- `subj` (`int32[]`) – Symmetric matrix variable index.
- `subk` (`int32[]`) – Block row index.
- `subl` (`int32[]`) – Block column index.
- `valjkl` (`float[]`) – The numerical value associated with each block triplet.

Groups

Problem data - semidefinite, *Inspecting the task*

`Task.getbarcidx`

```
getbarcidx(idx,sub,weights) -> (j,num)
```

```
getbarcidx(idx) -> (j,num,sub,weights)
```

Obtains information about an element in $\overline{C}$.

Parameters

- `idx` (`int64`) – Index of the element for which information should be obtained. (input)
- `sub` (`int64[]`) – Elements appearing the weighted sum. (output)
- `weights` (`float[]`) – Weights of terms in the weighted sum. (output)

Return

- `j` (`int32`) – Row index in $\overline{C}$.
- `num` (`int64`) – Number of terms in the weighted sum.
- `sub` (`int64[]`) – Elements appearing the weighted sum.

- `weights` (`float[]`) – Weights of terms in the weighted sum.

Groups
Problem data - semidefinite, *Inspecting the task*

`Task.getbarcidxinfo`

```
getbarcidxinfo(idx) -> (num)
```

Obtains the number of terms in the weighted sum that forms a particular element in $\overline{C}$.

Parameters
`idx` (`int64`) – Index of the element for which information should be obtained. The value is an index of a symmetric sparse variable. (input)

Return
`num` (`int64`) – Number of terms that appear in the weighted sum that forms the requested element.

Groups
Problem data - semidefinite, *Inspecting the task*

`Task.getbarcidxj`

```
getbarcidxj(idx) -> (j)
```

Obtains the row index of an element in $\overline{C}$.

Parameters
`idx` (`int64`) – Index of the element for which information should be obtained. (input)

Return
`j` (`int32`) – Row index in $\overline{C}$.

Groups
Problem data - semidefinite, *Inspecting the task*

`Task.getbarcsparsity`

```
getbarcsparsity(idxj) -> (numnz)
```

```
getbarcsparsity() -> (numnz,idxj)
```

Internally only the nonzero elements of $\overline{C}$ are stored in a vector. This function is used to obtain the nonzero elements of $\overline{C}$ and their indexes in the internal vector representation (in `idx`). From the index detailed information about each nonzero $\overline{C}_j$ can be obtained using *`Task.getbarcidxinfo`* and *`Task.getbarcidx`*.

Parameters
`idxj` (`int64[]`) – Internal positions of the nonzeros elements in $\overline{C}$. (output)

Return
- `numnz` (`int64`) – Number of nonzero elements in $\overline{C}$.
- `idxj` (`int64[]`) – Internal positions of the nonzeros elements in $\overline{C}$.

Groups
Problem data - semidefinite, *Inspecting the task*

`Task.getbarsj`

```
getbarsj(whichsol,j,barsj)
```

```
getbarsj(whichsol,j) -> (barsj)
```

Obtains the dual solution for a semidefinite variable. Only the lower triangular part of $\overline{S}_j$ is returned because the matrix by construction is symmetric. The format is that the columns are stored sequentially in the natural order.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `j` (`int32`) – Index of the semidefinite variable. (input)
- `barsj` (`float[]`) – Value of $\overline{S}_j$. (output)

Return

`barsj` (`float[]`) – Value of $\overline{S}_j$.

Groups

Solution - semidefinite

`Task.getbarsslice`

```
getbarsslice(whichsol,first,last,slicesize,barsslice)
```

```
getbarsslice(whichsol,first,last,slicesize) -> (barsslice)
```

Obtains the dual solution for a sequence of semidefinite variables. The format is that matrices are stored sequentially, and in each matrix the columns are stored as in *Task.getbarsj*.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (`int32`) – Index of the first semidefinite variable in the slice. (input)
- `last` (`int32`) – Index of the last semidefinite variable in the slice plus one. (input)
- `slicesize` (`int64`) – Denotes the length of the array `barsslice`. (input)
- `barsslice` (`float[]`) – Dual solution values of symmetric matrix variables in the slice, stored sequentially. (output)

Return

`barsslice` (`float[]`) – Dual solution values of symmetric matrix variables in the slice, stored sequentially.

Groups

Solution - semidefinite

`Task.getbarvarname`

```
getbarvarname(i) -> (name)
```

Obtains the name of a semidefinite variable.

Parameters

`i` (`int32`) – Index of the variable. (input)

Return

`name` (`str`) – The requested name is copied to this buffer.

Groups

Names, *Inspecting the task*

`Task.getbarvarnameindex`

```
getbarvarnameindex(somename) -> (asgn,index)
```

Obtains the index of semidefinite variable from its name.

Parameters

`somename` (`str`) – The name of the variable. (input)

Return

- `asgn` (`int32`) – Non-zero if the name `somename` is assigned to some semidefinite variable.
- `index` (`int32`) – The index of a semidefinite variable with the name `somename` (if one exists).

Groups
Names, *Inspecting the task*

`Task.getbarvarnamelen`

```
getbarvarnamelen(i) -> (len)
```

Obtains the length of the name of a semidefinite variable.

Parameters
`i` (`int32`) – Index of the variable. (input)
Return
`len` (`int32`) – Returns the length of the indicated name.
Groups
Names, *Inspecting the task*

`Task.getbarxj`

```
getbarxj(whichsol,j,barxj)
```

```
getbarxj(whichsol,j) -> (barxj)
```

Obtains the primal solution for a semidefinite variable. Only the lower triangular part of $\overline{X}_j$ is returned because the matrix by construction is symmetric. The format is that the columns are stored sequentially in the natural order.

Parameters
- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `j` (`int32`) – Index of the semidefinite variable. (input)
- `barxj` (`float[]`) – Value of $\overline{X}_j$. (output)

Return
`barxj` (`float[]`) – Value of $\overline{X}_j$.
Groups
Solution - semidefinite

`Task.getbarxslice`

```
getbarxslice(whichsol,first,last,slicesize,barxslice)
```

```
getbarxslice(whichsol,first,last,slicesize) -> (barxslice)
```

Obtains the primal solution for a sequence of semidefinite variables. The format is that matrices are stored sequentially, and in each matrix the columns are stored as in *Task.getbarxj*.

Parameters
- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (`int32`) – Index of the first semidefinite variable in the slice. (input)
- `last` (`int32`) – Index of the last semidefinite variable in the slice plus one. (input)
- `slicesize` (`int64`) – Denotes the length of the array `barxslice`. (input)
- `barxslice` (`float[]`) – Solution values of symmetric matrix variables in the slice, stored sequentially. (output)

Return
`barxslice` (`float[]`) – Solution values of symmetric matrix variables in the slice, stored sequentially.
Groups
Solution - semidefinite

`Task.getc`

```
getc(c)
```

```
getc() -> (c)
```

Obtains all objective coefficients c.

Parameters
`c` (`float[]`) – Linear terms of the objective as a dense vector. The length is the number of variables. (output)

Return
`c` (`float[]`) – Linear terms of the objective as a dense vector. The length is the number of variables.

Groups
Problem data - linear part, *Inspecting the task*, *Problem data - variables*

`Task.getcfix`

```
getcfix() -> (cfix)
```

Obtains the fixed term in the objective.

Return
`cfix` (`float`) – Fixed term in the objective.

Groups
Problem data - linear part, *Inspecting the task*

`Task.getcj`

```
getcj(j) -> (cj)
```

Obtains one coefficient of c.

Parameters
`j` (`int32`) – Index of the variable for which the c coefficient should be obtained. (input)

Return
`cj` (`float`) – The value of c_j.

Groups
Problem data - linear part, *Inspecting the task*, *Problem data - variables*

`Task.getclist`

```
getclist(subj,c)
```

```
getclist(subj) -> (c)
```

Obtains a sequence of elements in c.

Parameters
- `subj` (`int32[]`) – A list of variable indexes. (input)
- `c` (`float[]`) – Linear terms of the requested list of the objective as a dense vector. (output)

Return
`c` (`float[]`) – Linear terms of the requested list of the objective as a dense vector.

Groups
Inspecting the task, *Problem data - linear part*

`Task.getconbound`

```
getconbound(i) -> (bk,bl,bu)
```

Obtains bound information for one constraint.

Parameters

`i` (`int32`) – Index of the constraint for which the bound information should be obtained. (input)

Return

- `bk` (*mosek.boundkey*) – Bound keys.
- `bl` (`float`) – Values for lower bounds.
- `bu` (`float`) – Values for upper bounds.

Groups

Problem data - linear part, *Inspecting the task*, *Problem data - bounds*, *Problem data - constraints*

`Task.getconboundslice`

```
getconboundslice(first,last,bk,bl,bu)
```

```
getconboundslice(first,last) -> (bk,bl,bu)
```

Obtains bounds information for a slice of the constraints.

Parameters

- `first` (`int32`) – First index in the sequence. (input)
- `last` (`int32`) – Last index plus 1 in the sequence. (input)
- `bk` (*mosek.boundkey*`[]`) – Bound keys. (output)
- `bl` (`float[]`) – Values for lower bounds. (output)
- `bu` (`float[]`) – Values for upper bounds. (output)

Return

- `bk` (*mosek.boundkey*`[]`) – Bound keys.
- `bl` (`float[]`) – Values for lower bounds.
- `bu` (`float[]`) – Values for upper bounds.

Groups

Problem data - linear part, *Inspecting the task*, *Problem data - bounds*, *Problem data - constraints*

~~`Task.getcone`~~ *Deprecated*

```
getcone(k,submem) -> (ct,conepar,nummem)
```

```
getcone(k) -> (ct,conepar,nummem,submem)
```

NOTE: This interface to conic optimization is deprecated and will be removed in a future major release. Conic problems should be specified using the affine conic constraints interface (ACC), see Sec. 6.2 for details.

Parameters

- `k` (`int32`) – Index of the cone. (input)
- `submem` (`int32[]`) – Variable subscripts of the members in the cone. (output)

Return

- `ct` (*mosek.conetype*) – Specifies the type of the cone.
- `conepar` (`float`) – For the power cone it denotes the exponent alpha. For other cone types it is unused and can be set to 0.

- `nummem` (`int32`) – Number of member variables in the cone.
- `submem` (`int32[]`) – Variable subscripts of the members in the cone.

Groups
Inspecting the task, *Problem data - cones (deprecated)*

~~`Task.getconeinfo`~~ *Deprecated*

```
getconeinfo(k) -> (ct,conepar,nummem)
```

NOTE: This interface to conic optimization is deprecated and will be removed in a future major release. Conic problems should be specified using the affine conic constraints interface (ACC), see Sec. 6.2 for details.

Parameters
`k` (`int32`) – Index of the cone. (input)

Return
- `ct` (*`mosek.conetype`*) – Specifies the type of the cone.
- `conepar` (`float`) – For the power cone it denotes the exponent alpha. For other cone types it is unused and can be set to 0.
- `nummem` (`int32`) – Number of member variables in the cone.

Groups
Inspecting the task, *Problem data - cones (deprecated)*

~~`Task.getconename`~~ *Deprecated*

```
getconename(i) -> (name)
```

NOTE: This interface to conic optimization is deprecated and will be removed in a future major release. Conic problems should be specified using the affine conic constraints interface (ACC), see Sec. 6.2 for details.

Parameters
`i` (`int32`) – Index of the cone. (input)

Return
`name` (`str`) – The required name.

Groups
Names, *Problem data - cones (deprecated)*, *Inspecting the task*

~~`Task.getconenameindex`~~ *Deprecated*

```
getconenameindex(somename) -> (asgn,index)
```

NOTE: This interface to conic optimization is deprecated and will be removed in a future major release. Conic problems should be specified using the affine conic constraints interface (ACC), see Sec. 6.2 for details.

Checks whether the name `somename` has been assigned to any cone. If it has been assigned to a cone, then the index of the cone is reported.

Parameters
`somename` (`str`) – The name which should be checked. (input)

Return
- `asgn` (`int32`) – Is non-zero if the name `somename` is assigned to some cone.
- `index` (`int32`) – If the name `somename` is assigned to some cone, then `index` is the index of the cone.

Groups
Names, *Problem data - cones (deprecated)*, *Inspecting the task*

~~`Task.getconenamelen`~~ *Deprecated*

```
getconenamelen(i) -> (len)
```

NOTE: This interface to conic optimization is deprecated and will be removed in a future major release. Conic problems should be specified using the affine conic constraints interface (ACC), see Sec. 6.2 for details.

Parameters
`i` (`int32`) – Index of the cone. (input)

Return
`len` (`int32`) – Returns the length of the indicated name.

Groups
Names, *Problem data - cones (deprecated)*, *Inspecting the task*

`Task.getconname`

```
getconname(i) -> (name)
```

Obtains the name of a constraint.

Parameters
`i` (`int32`) – Index of the constraint. (input)

Return
`name` (`str`) – The required name.

Groups
Names, *Problem data - linear part*, *Problem data - constraints*, *Inspecting the task*

`Task.getconnameindex`

```
getconnameindex(somename) -> (asgn,index)
```

Checks whether the name `somename` has been assigned to any constraint. If so, the index of the constraint is reported.

Parameters
`somename` (`str`) – The name which should be checked. (input)

Return
- `asgn` (`int32`) – Is non-zero if the name `somename` is assigned to some constraint.
- `index` (`int32`) – If the name `somename` is assigned to a constraint, then `index` is the index of the constraint.

Groups
Names, *Problem data - linear part*, *Problem data - constraints*, *Inspecting the task*

`Task.getconnamelen`

```
getconnamelen(i) -> (len)
```

Obtains the length of the name of a constraint.

Parameters
`i` (`int32`) – Index of the constraint. (input)

Return
`len` (`int32`) – Returns the length of the indicated name.

Groups
Names, *Problem data - linear part*, *Problem data - constraints*, *Inspecting the task*

`Task.getcslice`

```
getcslice(first,last,c)
```

```
getcslice(first,last) -> (c)
```

Obtains a sequence of elements in c.

Parameters

- `first` (`int32`) – First index in the sequence. (input)
- `last` (`int32`) – Last index plus 1 in the sequence. (input)
- `c` (`float[]`) – Linear terms of the requested slice of the objective as a dense vector. The length is `last-first`. (output)

Return

`c` (`float[]`) – Linear terms of the requested slice of the objective as a dense vector. The length is `last-first`.

Groups

Inspecting the task, *Problem data - linear part*

`Task.getdimbarvarj`

```
getdimbarvarj(j) -> (dimbarvarj)
```

Obtains the dimension of a symmetric matrix variable.

Parameters

`j` (`int32`) – Index of the semidefinite variable whose dimension is requested. (input)

Return

`dimbarvarj` (`int32`) – The dimension of the j-th semidefinite variable.

Groups

Inspecting the task, *Problem data - semidefinite*

`Task.getdjcafeidxlist`

```
getdjcafeidxlist(djcidx,afeidxlist)
```

```
getdjcafeidxlist(djcidx) -> (afeidxlist)
```

Obtains the list of affine expression indexes in a disjunctive constraint.

Parameters

- `djcidx` (`int64`) – Index of the disjunctive constraint. (input)
- `afeidxlist` (`int64[]`) – List of affine expression indexes. (output)

Return

`afeidxlist` (`int64[]`) – List of affine expression indexes.

Groups

Problem data - disjunctive constraints, *Inspecting the task*

`Task.getdjcb`

```
getdjcb(djcidx,b)
```

```
getdjcb(djcidx) -> (b)
```

Obtains the optional constant term vector of a disjunctive constraint.

Parameters

- `djcidx` (`int64`) – Index of the disjunctive constraint. (input)
- `b` (`float[]`) – The vector b. (output)

Return

`b` (`float[]`) – The vector b.

Groups

Problem data - disjunctive constraints, *Inspecting the task*

`Task.getdjcdomainidxlist`

```
getdjcdomainidxlist(djcidx,domidxlist)
```

```
getdjcdomainidxlist(djcidx) -> (domidxlist)
```

Obtains the list of domain indexes in a disjunctive constraint.

Parameters

- `djcidx` (`int64`) – Index of the disjunctive constraint. (input)
- `domidxlist` (`int64[]`) – List of term sizes. (output)

Return

`domidxlist` (`int64[]`) – List of term sizes.

Groups

Problem data - disjunctive constraints, *Inspecting the task*

`Task.getdjcname`

```
getdjcname(djcidx) -> (name)
```

Obtains the name of a disjunctive constraint.

Parameters

`djcidx` (`int64`) – Index of a disjunctive constraint. (input)

Return

`name` (`str`) – Returns the required name.

Groups

Names, *Problem data - disjunctive constraints*, *Inspecting the task*

`Task.getdjcnamelen`

```
getdjcnamelen(djcidx) -> (len)
```

Obtains the length of the name of a disjunctive constraint.

Parameters

`djcidx` (`int64`) – Index of a disjunctive constraint. (input)

Return

`len` (`int32`) – Returns the length of the indicated name.

Groups

Names, *Problem data - disjunctive constraints*, *Inspecting the task*

`Task.getdjcnumafe`

```
getdjcnumafe(djcidx) -> (numafe)
```

Obtains the number of affine expressions in the disjunctive constraint.

Parameters

`djcidx` (`int64`) – Index of the disjunctive constraint. (input)

Return

`numafe` (`int64`) – Number of affine expressions in the disjunctive constraint.

Groups

Problem data - disjunctive constraints, *Inspecting the task*

`Task.getdjcnumafetot`

```
getdjcnumafetot() -> (numafetot)
```

Obtains the total number of affine expressions in all disjunctive constraints.

Return
`numafetot` (`int64`) – Number of affine expressions in all disjunctive constraints.

Groups
Problem data - disjunctive constraints, *Inspecting the task*

`Task.getdjcnumdomain`

```
getdjcnumdomain(djcidx) -> (numdomain)
```

Obtains the number of domains in the disjunctive constraint.

Parameters
`djcidx` (`int64`) – Index of the disjunctive constraint. (input)

Return
`numdomain` (`int64`) – Number of domains in the disjunctive constraint.

Groups
Problem data - disjunctive constraints, *Inspecting the task*

`Task.getdjcnumdomaintot`

```
getdjcnumdomaintot() -> (numdomaintot)
```

Obtains the total number of domains in all disjunctive constraints.

Return
`numdomaintot` (`int64`) – Number of domains in all disjunctive constraints.

Groups
Problem data - disjunctive constraints, *Inspecting the task*

`Task.getdjcnumterm`

```
getdjcnumterm(djcidx) -> (numterm)
```

Obtains the number terms in the disjunctive constraint.

Parameters
`djcidx` (`int64`) – Index of the disjunctive constraint. (input)

Return
`numterm` (`int64`) – Number of terms in the disjunctive constraint.

Groups
Problem data - disjunctive constraints, *Inspecting the task*

`Task.getdjcnumtermtot`

```
getdjcnumtermtot() -> (numtermtot)
```

Obtains the total number of terms in all disjunctive constraints.

Return
`numtermtot` (`int64`) – Total number of terms in all disjunctive constraints.

Groups
Problem data - disjunctive constraints, *Inspecting the task*

`Task.getdjcs`

```
getdjcs(domidxlist,
        afeidxlist,
        b,
        termsizelist,
        numterms)
```

```
getdjcs() ->
       (domidxlist,
        afeidxlist,
        b,
        termsizelist,
        numterms)
```

Obtains full data of all disjunctive constraints. The output arrays must have minimal lengths determined by the following methods: `domainidxlist` by *Task.getdjcnumdomaintot*, `afeidxlist` and `b` by *Task.getdjcnumafetot*, `termsizelist` by *Task.getdjcnumtermtot* and `numterms` by *Task.getnumdomain*.

Parameters

- `domidxlist` (`int64[]`) – The concatenation of index lists of domains appearing in all disjunctive constraints. (output)
- `afeidxlist` (`int64[]`) – The concatenation of index lists of affine expressions appearing in all disjunctive constraints. (output)
- `b` (`float[]`) – The concatenation of vectors b appearing in all disjunctive constraints. (output)
- `termsizelist` (`int64[]`) – The concatenation of lists of term sizes appearing in all disjunctive constraints. (output)
- `numterms` (`int64[]`) – The number of terms in each of the disjunctive constraints. (output)

Return

- `domidxlist` (`int64[]`) – The concatenation of index lists of domains appearing in all disjunctive constraints.
- `afeidxlist` (`int64[]`) – The concatenation of index lists of affine expressions appearing in all disjunctive constraints.
- `b` (`float[]`) – The concatenation of vectors b appearing in all disjunctive constraints.
- `termsizelist` (`int64[]`) – The concatenation of lists of term sizes appearing in all disjunctive constraints.
- `numterms` (`int64[]`) – The number of terms in each of the disjunctive constraints.

Groups

Problem data - disjunctive constraints, *Inspecting the task*

`Task.getdjctermsizelist`

```
getdjctermsizelist(djcidx,termsizelist)
```

```
getdjctermsizelist(djcidx) -> (termsizelist)
```

Obtains the list of term sizes in a disjunctive constraint.

Parameters

- `djcidx` (`int64`) – Index of the disjunctive constraint. (input)
- `termsizelist` (`int64[]`) – List of term sizes. (output)

Return

`termsizelist` (`int64[]`) – List of term sizes.

Groups

Problem data - disjunctive constraints, *Inspecting the task*

`Task.getdomainn`

```
getdomainn(domidx) -> (n)
```

Obtains the dimension of the domain.

Parameters
`domidx` (`int64`) – Index of the domain. (input)

Return
`n` (`int64`) – Dimension of the domain.

Groups
Problem data - domain, *Inspecting the task*

`Task.getdomainname`

```
getdomainname(domidx) -> (name)
```

Obtains the name of a domain.

Parameters
`domidx` (`int64`) – Index of a domain. (input)

Return
`name` (`str`) – Returns the required name.

Groups
Names, *Problem data - domain*, *Inspecting the task*

`Task.getdomainnamelen`

```
getdomainnamelen(domidx) -> (len)
```

Obtains the length of the name of a domain.

Parameters
`domidx` (`int64`) – Index of a domain. (input)

Return
`len` (`int32`) – Returns the length of the indicated name.

Groups
Names, *Problem data - domain*, *Inspecting the task*

`Task.getdomaintype`

```
getdomaintype(domidx) -> (domtype)
```

Returns the type of the domain.

Parameters
`domidx` (`int64`) – Index of the domain. (input)

Return
`domtype` (*`mosek.domaintype`*) – The type of the domain.

Groups
Problem data - domain, *Inspecting the task*

`Task.getdouinf`

```
getdouinf(whichdinf) -> (dvalue)
```

Obtains a double information item from the task information database.

Parameters
`whichdinf` (*`mosek.dinfitem`*) – Specifies a double information item. (input)

Return
`dvalue` (`float`) – The value of the required double information item.

Groups
Information items and statistics

`Task.getdouparam`

```
getdouparam(param) -> (parvalue)
```

Obtains the value of a double parameter.

Parameters
`param` (*mosek.dparam*) – Which parameter. (input)

Return
`parvalue` (`float`) – Parameter value.

Groups
Parameters

`Task.getdualobj`

```
getdualobj(whichsol) -> (dualobj)
```

Computes the dual objective value associated with the solution. Note that if the solution is a primal infeasibility certificate, then the fixed term in the objective value is not included.

Moreover, since there is no dual solution associated with an integer solution, an error will be reported if the dual objective value is requested for the integer solution.

Parameters
`whichsol` (*mosek.soltype*) – Selects a solution. (input)

Return
`dualobj` (`float`) – Objective value corresponding to the dual solution.

Groups
Solution information, *Solution - dual*

`Task.getdualproblem`

```
getdualproblem() -> (dualtask)
```

Returns the dual problem as a task. The dual computed by this function is intended for demonstration, educational and debugging purposes. It closely follows the dual form introduced in the documentation, but need not exactly reflect dualization taking place internally in the solver.

The function attempts to detect sparse LMIs and remove redundant linear constraints, trying to formulate the most efficient dual LMI (i.e. SDP in primal form), unless *iparam.getdual_convert_lmis* is turned off.

Problems with quadratic objective or constraints are not supported. Integer variables and disjunctive constraints are ignored and only a warning is issued.

Return
`dualtask` (*mosek.Task*) – A new task containing the dualized problem.

Groups
Environment and task management

`Task.getdualsolutionnorms`

```
getdualsolutionnorms(whichsol) ->
                     (nrmy,
                      nrmslc,
                      nrmsuc,
                      nrmslx,
                      nrmsux,
                      nrmsnx,
                      nrmbars)
```

Compute norms of the dual solution.

Parameters
`whichsol` (*mosek.soltype*) – Selects a solution. (input)

Return

- `nrmy` (`float`) – The norm of the y vector.
- `nrmslc` (`float`) – The norm of the s_l^c vector.
- `nrmsuc` (`float`) – The norm of the s_u^c vector.
- `nrmslx` (`float`) – The norm of the s_l^x vector.
- `nrmsux` (`float`) – The norm of the s_u^x vector.
- `nrmsnx` (`float`) – The norm of the s_n^x vector.
- `nrmbars` (`float`) – The norm of the $\overline{S}$ vector.

Groups

Solution information

`Task.getdviolacc`

```
getdviolacc(whichsol,accidxlist,viol)
```

```
getdviolacc(whichsol,accidxlist) -> (viol)
```

Let $(s_n^x)^*$ be the value of variable (s_n^x) for the specified solution. For simplicity let us assume that s_n^x is a member of a quadratic cone, then the violation is computed as follows

$$\begin{cases} \max(0, (\|s_n^x\|_{2:n}^* - (s_n^x)_1^*)/\sqrt{2}, & (s_n^x)^* \geq -\|(s_n^x)_{2:n}^*\|, \\ \|(s_n^x)^*\|, & \text{otherwise.} \end{cases}$$

Both when the solution is a certificate of primal infeasibility or when it is a dual feasible solution the violation should be small.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `accidxlist` (`int64[]`) – An array of indexes of conic constraints. (input)
- `viol` (`float[]`) – `viol[k]` is the violation of the dual solution associated with the conic constraint `sub[k]`. (output)

Return

`viol` (`float[]`) – `viol[k]` is the violation of the dual solution associated with the conic constraint `sub[k]`.

Groups

Solution information

`Task.getdviolbarvar`

```
getdviolbarvar(whichsol,sub,viol)
```

```
getdviolbarvar(whichsol,sub) -> (viol)
```

Let $(\overline{S}_j)^*$ be the value of variable $\overline{S}_j$ for the specified solution. Then the dual violation of the solution associated with variable $\overline{S}_j$ is given by

$$\max(-\lambda_{\min}(\overline{S}_j),\ 0.0).$$

Both when the solution is a certificate of primal infeasibility and when it is dual feasible solution the violation should be small.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `sub` (`int32[]`) – An array of indexes of $\overline{X}$ variables. (input)
- `viol` (`float[]`) – `viol[k]` is the violation of the solution for the constraint $\overline{S}_{\texttt{sub}[k]} \in \mathcal{S}_+$. (output)

Return

`viol` (`float[]`) – `viol[k]` is the violation of the solution for the constraint $\overline{S}_{\texttt{sub}[k]} \in \mathcal{S}_+$.

Groups

Solution information

`Task.getdviolcon`

```
getdviolcon(whichsol,sub,viol)
```

```
getdviolcon(whichsol,sub) -> (viol)
```

The violation of the dual solution associated with the i-th constraint is computed as follows

$$\max(\rho((s_l^c)_i^*, (b_l^c)_i),\ \rho((s_u^c)_i^*, -(b_u^c)_i),\ |-y_i + (s_l^c)_i^* - (s_u^c)_i^*|)$$

where

$$\rho(x,l) = \begin{cases} -x, & l > -\infty, \\ |x|, & \text{otherwise.} \end{cases}$$

Both when the solution is a certificate of primal infeasibility or it is a dual feasible solution the violation should be small.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `sub` (`int32[]`) – An array of indexes of constraints. (input)
- `viol` (`float[]`) – `viol[k]` is the violation of dual solution associated with the constraint `sub[k]`. (output)

Return

`viol` (`float[]`) – `viol[k]` is the violation of dual solution associated with the constraint `sub[k]`.

Groups

Solution information

~~`Task.getdviolcones`~~ *Deprecated*

```
getdviolcones(whichsol,sub,viol)
```

```
getdviolcones(whichsol,sub) -> (viol)
```

NOTE: This interface to conic optimization is deprecated and will be removed in a future major release. Conic problems should be specified using the affine conic constraints interface (ACC), see Sec. 6.2 for details.

Let $(s_n^x)^*$ be the value of variable (s_n^x) for the specified solution. For simplicity let us assume that s_n^x is a member of a quadratic cone, then the violation is computed as follows

$$\begin{cases} \max(0, (\|s_n^x\|_{2:n}^* - (s_n^x)_1^*)/\sqrt{2}, & (s_n^x)^* \geq -\|(s_n^x)_{2:n}^*\|, \\ \|(s_n^x)^*\|, & \text{otherwise.} \end{cases}$$

Both when the solution is a certificate of primal infeasibility or when it is a dual feasible solution the violation should be small.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `sub` (`int32[]`) – An array of indexes of conic constraints. (input)
- `viol` (`float[]`) – `viol[k]` is the violation of the dual solution associated with the conic constraint `sub[k]`. (output)

Return

`viol` (`float[]`) – `viol[k]` is the violation of the dual solution associated with the conic constraint `sub[k]`.

Groups

Solution information

`Task.getdviolvar`

```
getdviolvar(whichsol,sub,viol)
```

```
getdviolvar(whichsol,sub) -> (viol)
```

The violation of the dual solution associated with the j-th variable is computed as follows

$$\max\left(\rho((s_l^x)_j^*,(b_l^x)_j),\ \rho((s_u^x)_j^*,-(b_u^x)_j),\ \left|\sum_{i=0}^{\texttt{numcon}-1} a_{ij}y_i+(s_l^x)_j^*-(s_u^x)_j^*-\tau c_j\right|\right)$$

where

$$\rho(x,l)=\begin{cases}-x, & l>-\infty,\\ |x|, & \text{otherwise}\end{cases}$$

and $\tau=0$ if the solution is a certificate of primal infeasibility and $\tau=1$ otherwise. The formula for computing the violation is only shown for the linear case but is generalized appropriately for the more general problems. Both when the solution is a certificate of primal infeasibility or when it is a dual feasible solution the violation should be small.

Parameters

- `whichsol` (*`mosek.soltype`*) – Selects a solution. (input)
- `sub` (`int32[]`) – An array of indexes of x variables. (input)
- `viol` (`float[]`) – `viol[k]` is the violation of dual solution associated with the variable `sub[k]`. (output)

Return

`viol` (`float[]`) – `viol[k]` is the violation of dual solution associated with the variable `sub[k]`.

Groups

Solution information

`Task.getinfeasiblesubproblem`

```
getinfeasiblesubproblem(whichsol) -> (inftask)
```

Given the solution is a certificate of primal or dual infeasibility then a primal or dual infeasible subproblem is obtained respectively. The subproblem tends to be much smaller than the original problem and hence it is easier to locate the infeasibility inspecting the subproblem than the original problem.

For the procedure to be useful it is important to assign meaningful names to constraints, variables etc. in the original task because those names will be duplicated in the subproblem.

The function is only applicable to linear and conic quadratic optimization problems.

For more information see Sec. 8.3 and Sec. 14.2.

Parameters

`whichsol` (*`mosek.soltype`*) – Which solution to use when determining the infeasible subproblem. (input)

Return

`inftask` (*`mosek.Task`*) – A new task containing the infeasible subproblem.

Groups

Infeasibility diagnostic

`Task.getintinf`

```
getintinf(whichiinf) -> (ivalue)
```

Obtains an integer information item from the task information database.

Parameters
`whichiinf` (*mosek.iinfitem*) – Specifies an integer information item. (input)

Return
`ivalue` (`int32`) – The value of the required integer information item.

Groups
Information items and statistics

`Task.getintparam`

```
getintparam(param) -> (parvalue)
```

Obtains the value of an integer parameter.

Parameters
`param` (*mosek.iparam*) – Which parameter. (input)

Return
`parvalue` (`int32`) – Parameter value.

Groups
Parameters

`Task.getlenbarvarj`

```
getlenbarvarj(j) -> (lenbarvarj)
```

Obtains the length of the j-th semidefinite variable i.e. the number of elements in the lower triangular part.

Parameters
`j` (`int32`) – Index of the semidefinite variable whose length if requested. (input)

Return
`lenbarvarj` (`int64`) – Number of scalar elements in the lower triangular part of the semidefinite variable.

Groups
Inspecting the task, *Problem data - semidefinite*

`Task.getlintinf`

```
getlintinf(whichliinf) -> (ivalue)
```

Obtains a long integer information item from the task information database.

Parameters
`whichliinf` (*mosek.liinfitem*) – Specifies a long information item. (input)

Return
`ivalue` (`int64`) – The value of the required long integer information item.

Groups
Information items and statistics

`Task.getlintparam`

```
getlintparam(param) -> (parvalue)
```

Obtains the value of an integer parameter.

Parameters
`param` (*mosek.iparam*) – Which parameter. (input)

Return
`parvalue` (`int64`) – Parameter value.

Groups
Parameters

`Task.getmaxnumanz`

```
getmaxnumanz() -> (maxnumanz)
```

Obtains number of preallocated non-zeros in A. When this number of non-zeros is reached **MOSEK** will automatically allocate more space for A.

Return
`maxnumanz` (`int64`) – Number of preallocated non-zero linear matrix elements.

Groups
Inspecting the task, *Problem data - linear part*

`Task.getmaxnumbarvar`

```
getmaxnumbarvar() -> (maxnumbarvar)
```

Obtains maximum number of symmetric matrix variables for which space is currently preallocated.

Return
`maxnumbarvar` (`int32`) – Maximum number of symmetric matrix variables for which space is currently preallocated.

Groups
Inspecting the task, *Problem data - semidefinite*

`Task.getmaxnumcon`

```
getmaxnumcon() -> (maxnumcon)
```

Obtains the number of preallocated constraints in the optimization task. When this number of constraints is reached **MOSEK** will automatically allocate more space for constraints.

Return
`maxnumcon` (`int32`) – Number of preallocated constraints in the optimization task.

Groups
Inspecting the task, *Problem data - linear part*, *Problem data - constraints*

~~`Task.getmaxnumcone`~~ *Deprecated*

```
getmaxnumcone() -> (maxnumcone)
```

NOTE: This interface to conic optimization is deprecated and will be removed in a future major release. Conic problems should be specified using the affine conic constraints interface (ACC), see Sec. 6.2 for details.

Obtains the number of preallocated cones in the optimization task. When this number of cones is reached **MOSEK** will automatically allocate space for more cones.

Return
`maxnumcone` (`int32`) – Number of preallocated conic constraints in the optimization task.

Groups
Inspecting the task, *Problem data - cones (deprecated)*

`Task.getmaxnumqnz`

```
getmaxnumqnz() -> (maxnumqnz)
```

Obtains the number of preallocated non-zeros for Q (both objective and constraints). When this number of non-zeros is reached **MOSEK** will automatically allocate more space for Q.

Return

`maxnumqnz` (`int64`) – Number of non-zero elements preallocated in quadratic coefficient matrices.

Groups

Inspecting the task, *Problem data - quadratic part*

`Task.getmaxnumvar`

```
getmaxnumvar() -> (maxnumvar)
```

Obtains the number of preallocated variables in the optimization task. When this number of variables is reached **MOSEK** will automatically allocate more space for variables.

Return

`maxnumvar` (`int32`) – Number of preallocated variables in the optimization task.

Groups

Inspecting the task, *Problem data - linear part*, *Problem data - variables*

`Task.getmemusage`

```
getmemusage() -> (meminuse,maxmemuse)
```

Obtains information about the amount of memory used by a task.

Return

- `meminuse` (`int64`) – Amount of memory currently used by the `task`.
- `maxmemuse` (`int64`) – Maximum amount of memory used by the `task` until now.

Groups

System, memory and debugging

`Task.getnumacc`

```
getnumacc() -> (num)
```

Obtains the number of affine conic constraints.

Return

`num` (`int64`) – The number of affine conic constraints.

Groups

Problem data - affine conic constraints, *Inspecting the task*

`Task.getnumafe`

```
getnumafe() -> (numafe)
```

Obtains the number of affine expressions.

Return

`numafe` (`int64`) – Number of affine expressions.

Groups

Problem data - affine expressions, *Inspecting the task*

`Task.getnumanz`

```
getnumanz() -> (numanz)
```

Obtains the number of non-zeros in A.

Return
`numanz` (`int64`) – Number of non-zero elements in the linear constraint matrix.

Groups
Inspecting the task, *Problem data - linear part*

`Task.getnumbarablocktriplets`

```
getnumbarablocktriplets() -> (num)
```

Obtains an upper bound on the number of elements in the block triplet form of $\overline{A}$.

Return
`num` (`int64`) – An upper bound on the number of elements in the block triplet form of $\overline{A}$.

Groups
Problem data - semidefinite, *Inspecting the task*

`Task.getnumbaranz`

```
getnumbaranz() -> (nz)
```

Get the number of nonzero elements in $\overline{A}$.

Return
`nz` (`int64`) – The number of nonzero block elements in $\overline{A}$ i.e. the number of $\overline{A}_{ij}$ elements that are nonzero.

Groups
Problem data - semidefinite, *Inspecting the task*

`Task.getnumbarcblocktriplets`

```
getnumbarcblocktriplets() -> (num)
```

Obtains an upper bound on the number of elements in the block triplet form of $\overline{C}$.

Return
`num` (`int64`) – An upper bound on the number of elements in the block triplet form of $\overline{C}$.

Groups
Problem data - semidefinite, *Inspecting the task*

`Task.getnumbarcnz`

```
getnumbarcnz() -> (nz)
```

Obtains the number of nonzero elements in $\overline{C}$.

Return
`nz` (`int64`) – The number of nonzeros in $\overline{C}$ i.e. the number of elements $\overline{C}_j$ that are nonzero.

Groups
Problem data - semidefinite, *Inspecting the task*

`Task.getnumbarvar`

```
getnumbarvar() -> (numbarvar)
```

Obtains the number of semidefinite variables.

Return
`numbarvar` (`int32`) – Number of semidefinite variables in the problem.

Groups
Inspecting the task, *Problem data - semidefinite*

`Task.getnumcon`

```
getnumcon() -> (numcon)
```

Obtains the number of constraints.

Return
`numcon` (`int32`) – Number of constraints.

Groups
Problem data - linear part, *Problem data - constraints*, *Inspecting the task*

~~`Task.getnumcone`~~ *Deprecated*

```
getnumcone() -> (numcone)
```

NOTE: This interface to conic optimization is deprecated and will be removed in a future major release. Conic problems should be specified using the affine conic constraints interface (ACC), see Sec. 6.2 for details.

Return
`numcone` (`int32`) – Number of conic constraints.

Groups
Problem data - cones (deprecated), *Inspecting the task*

~~`Task.getnumconemem`~~ *Deprecated*

```
getnumconemem(k) -> (nummem)
```

NOTE: This interface to conic optimization is deprecated and will be removed in a future major release. Conic problems should be specified using the affine conic constraints interface (ACC), see Sec. 6.2 for details.

Parameters
`k` (`int32`) – Index of the cone. (input)

Return
`nummem` (`int32`) – Number of member variables in the cone.

Groups
Problem data - cones (deprecated), *Inspecting the task*

`Task.getnumdjc`

```
getnumdjc() -> (num)
```

Obtains the number of disjunctive constraints.

Return
`num` (`int64`) – The number of disjunctive constraints.

Groups
Problem data - disjunctive constraints, *Inspecting the task*

`Task.getnumdomain`

```
getnumdomain() -> (numdomain)
```

Obtain the number of domains defined.

Return
`numdomain` (`int64`) – Number of domains in the task.

Groups
Inspecting the task, *Problem data - domain*

`Task.getnumintvar`

```
getnumintvar() -> (numintvar)
```

Obtains the number of integer-constrained variables.

Return
`numintvar` (`int32`) – Number of integer variables.

Groups
Inspecting the task, *Problem data - variables*

`Task.getnumparam`

```
getnumparam(partype) -> (numparam)
```

Obtains the number of parameters of a given type.

Parameters
`partype` (*mosek.parametertype*) – Parameter type. (input)

Return
`numparam` (`int32`) – The number of parameters of type `partype`.

Groups
Inspecting the task, *Parameters*

`Task.getnumqconknz`

```
getnumqconknz(k) -> (numqcnz)
```

Obtains the number of non-zero quadratic terms in a constraint.

Parameters
`k` (`int32`) – Index of the constraint for which the number quadratic terms should be obtained. (input)

Return
`numqcnz` (`int64`) – Number of quadratic terms.

Groups
Inspecting the task, *Problem data - constraints*, *Problem data - quadratic part*

`Task.getnumqobjnz`

```
getnumqobjnz() -> (numqonz)
```

Obtains the number of non-zero quadratic terms in the objective.

Return
`numqonz` (`int64`) – Number of non-zero elements in the quadratic objective terms.

Groups
Inspecting the task, *Problem data - quadratic part*

`Task.getnumsymmat`

```
getnumsymmat() -> (num)
```

Obtains the number of symmetric matrices stored in the vector E.

Return
`num` (`int64`) – The number of symmetric sparse matrices.

Groups
Problem data - semidefinite, *Inspecting the task*

`Task.getnumvar`

```
getnumvar() -> (numvar)
```

Obtains the number of variables.

Return
`numvar` (`int32`) – Number of variables.

Groups
Inspecting the task, *Problem data - variables*

`Task.getobjname`

```
getobjname() -> (objname)
```

Obtains the name assigned to the objective function.

Return
`objname` (`str`) – Assigned the objective name.

Groups
Inspecting the task, *Names*

`Task.getobjnamelen`

```
getobjnamelen() -> (len)
```

Obtains the length of the name assigned to the objective function.

Return
`len` (`int32`) – Assigned the length of the objective name.

Groups
Inspecting the task, *Names*

`Task.getobjsense`

```
getobjsense() -> (sense)
```

Gets the objective sense of the task.

Return
`sense` (*`mosek.objsense`*) – The returned objective sense.

Groups
Problem data - linear part

`Task.getpowerdomainalpha`

```
getpowerdomainalpha(domidx,alpha)
```

```
getpowerdomainalpha(domidx) -> (alpha)
```

Obtains the exponent vector α of a primal or dual power cone domain.

Parameters
- `domidx` (`int64`) – Index of the domain. (input)
- `alpha` (`float[]`) – The vector α. (output)

Return
`alpha` (`float[]`) – The vector α.

Groups
Problem data - domain, *Inspecting the task*

`Task.getpowerdomaininfo`

```
getpowerdomaininfo(domidx) -> (n,nleft)
```

Obtains structural information about a primal or dual power cone domain.

Parameters
`domidx` (`int64`) – Index of the domain. (input)

Return
- `n` (`int64`) – Dimension of the domain.
- `nleft` (`int64`) – Number of variables on the left hand side.

Groups
Problem data - domain, *Inspecting the task*

`Task.getprimalobj`

```
getprimalobj(whichsol) -> (primalobj)
```

Computes the primal objective value for the desired solution. Note that if the solution is an infeasibility certificate, then the fixed term in the objective is not included.

Parameters
`whichsol` (*`mosek.soltype`*) – Selects a solution. (input)

Return
`primalobj` (`float`) – Objective value corresponding to the primal solution.

Groups
Solution information, *Solution - primal*

`Task.getprimalsolutionnorms`

```
getprimalsolutionnorms(whichsol) -> (nrmxc,nrmxx,nrmbarx)
```

Compute norms of the primal solution.

Parameters
`whichsol` (*`mosek.soltype`*) – Selects a solution. (input)

Return
- `nrmxc` (`float`) – The norm of the x^c vector.
- `nrmxx` (`float`) – The norm of the x vector.
- `nrmbarx` (`float`) – The norm of the $\overline{X}$ vector.

Groups
Solution information

`Task.getprobtype`

```
getprobtype() -> (probtype)
```

Obtains the problem type.

Return
`probtype` (*`mosek.problemtype`*) – The problem type.

Groups
Inspecting the task

`Task.getprosta`

```
getprosta(whichsol) -> (problemsta)
```

Obtains the problem status.

Parameters
`whichsol` (*mosek.soltype*) – Selects a solution. (input)

Return
`problemsta` (*mosek.prosta*) – Problem status.

Groups
Solution information

`Task.getpviolacc`

```
getpviolacc(whichsol,accidxlist,viol)
```

```
getpviolacc(whichsol,accidxlist) -> (viol)
```

Computes the primal solution violation for a set of affine conic constraints. Let x^* be the value of the variable x for the specified solution. For simplicity let us assume that x is a member of a quadratic cone, then the violation is computed as follows

$$\begin{cases} \max(0, \|x_{2:n}\| - x_1)/\sqrt{2}, & x_1 \geq -\|x_{2:n}\|, \\ \|x\|, & \text{otherwise.} \end{cases}$$

Both when the solution is a certificate of dual infeasibility or when it is primal feasible the violation should be small.

Parameters
- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `accidxlist` (`int64[]`) – An array of indexes of conic constraints. (input)
- `viol` (`float[]`) – `viol[k]` is the violation of the solution associated with the affine conic constraint number `accidxlist[k]`. (output)

Return
`viol` (`float[]`) – `viol[k]` is the violation of the solution associated with the affine conic constraint number `accidxlist[k]`.

Groups
Solution information

`Task.getpviolbarvar`

```
getpviolbarvar(whichsol,sub,viol)
```

```
getpviolbarvar(whichsol,sub) -> (viol)
```

Computes the primal solution violation for a set of semidefinite variables. Let $(\overline{X}_j)^*$ be the value of the variable $\overline{X}_j$ for the specified solution. Then the primal violation of the solution associated with variable $\overline{X}_j$ is given by

$$\max(-\lambda_{\min}(\overline{X}_j),\ 0.0).$$

Both when the solution is a certificate of dual infeasibility or when it is primal feasible the violation should be small.

Parameters
- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `sub` (`int32[]`) – An array of indexes of $\overline{X}$ variables. (input)

- `viol` (`float[]`) – `viol[k]` is how much the solution violates the constraint $\overline{X}_{\texttt{sub}[k]} \in \mathcal{S}_+$. (output)

Return

`viol` (`float[]`) – `viol[k]` is how much the solution violates the constraint $\overline{X}_{\texttt{sub}[k]} \in \mathcal{S}_+$.

Groups

Solution information

`Task.getpviolcon`

```
getpviolcon(whichsol,sub,viol)
```

```
getpviolcon(whichsol,sub) -> (viol)
```

Computes the primal solution violation for a set of constraints. The primal violation of the solution associated with the i-th constraint is given by

$$\max(\tau l_i^c - (x_i^c)^*,\ (x_i^c)^* - \tau u_i^c),\ |\ \sum_{j=0}^{\texttt{numvar}-1} a_{ij} x_j^* - x_i^c|)$$

where $\tau = 0$ if the solution is a certificate of dual infeasibility and $\tau = 1$ otherwise. Both when the solution is a certificate of dual infeasibility and when it is primal feasible the violation should be small. The above formula applies for the linear case but is appropriately generalized in other cases.

Parameters

- `whichsol` (*`mosek.soltype`*) – Selects a solution. (input)
- `sub` (`int32[]`) – An array of indexes of constraints. (input)
- `viol` (`float[]`) – `viol[k]` is the violation associated with the solution for the constraint `sub[k]`. (output)

Return

`viol` (`float[]`) – `viol[k]` is the violation associated with the solution for the constraint `sub[k]`.

Groups

Solution information

~~`Task.getpviolcones`~~ *Deprecated*

```
getpviolcones(whichsol,sub,viol)
```

```
getpviolcones(whichsol,sub) -> (viol)
```

NOTE: This interface to conic optimization is deprecated and will be removed in a future major release. Conic problems should be specified using the affine conic constraints interface (ACC), see Sec. 6.2 for details.

Computes the primal solution violation for a set of conic constraints. Let x^* be the value of the variable x for the specified solution. For simplicity let us assume that x is a member of a quadratic cone, then the violation is computed as follows

$$\begin{cases} \max(0, \|x_{2:n}\| - x_1)/\sqrt{2}, & x_1 \geq -\|x_{2:n}\|, \\ \|x\|, & \text{otherwise.} \end{cases}$$

Both when the solution is a certificate of dual infeasibility or when it is primal feasible the violation should be small.

Parameters

- `whichsol` (*`mosek.soltype`*) – Selects a solution. (input)
- `sub` (`int32[]`) – An array of indexes of conic constraints. (input)

- `viol` (`float[]`) – `viol[k]` is the violation of the solution associated with the conic constraint number `sub[k]`. (output)

Return

`viol` (`float[]`) – `viol[k]` is the violation of the solution associated with the conic constraint number `sub[k]`.

Groups

Solution information

`Task.getpvioldjc`

```
getpvioldjc(whichsol,djcidxlist,viol)
```

```
getpvioldjc(whichsol,djcidxlist) -> (viol)
```

Computes the primal solution violation for a set of disjunctive constraints. For a single DJC the violation is defined as

$$\mathrm{viol}\left(\bigvee_{i=1}^{t}\bigwedge_{j=1}^{s_i} T_{i,j}\right) = \min_{i=1,\ldots,t}\left(\max_{j=1,\ldots,s_j}(\mathrm{viol}(T_{i,j}))\right)$$

where the violation of each simple term $T_{i,j}$ is defined as for an ordinary linear constraint.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `djcidxlist` (`int64[]`) – An array of indexes of disjunctive constraints. (input)
- `viol` (`float[]`) – `viol[k]` is the violation of the solution associated with the disjunctive constraint number `djcidxlist[k]`. (output)

Return

`viol` (`float[]`) – `viol[k]` is the violation of the solution associated with the disjunctive constraint number `djcidxlist[k]`.

Groups

Solution information

`Task.getpviolvar`

```
getpviolvar(whichsol,sub,viol)
```

```
getpviolvar(whichsol,sub) -> (viol)
```

Computes the primal solution violation associated to a set of variables. Let x_j^* be the value of x_j for the specified solution. Then the primal violation of the solution associated with variable x_j is given by

$$\max(\tau l_j^x - x_j^*,\ x_j^* - \tau u_j^x,\ 0).$$

where $\tau = 0$ if the solution is a certificate of dual infeasibility and $\tau = 1$ otherwise. Both when the solution is a certificate of dual infeasibility and when it is primal feasible the violation should be small.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `sub` (`int32[]`) – An array of indexes of x variables. (input)
- `viol` (`float[]`) – `viol[k]` is the violation associated with the solution for the variable $x_{\texttt{sub[k]}}$. (output)

Return

`viol` (`float[]`) – `viol[k]` is the violation associated with the solution for the variable $x_{\texttt{sub[k]}}$.

Groups

Solution information

`Task.getqconk`

```
getqconk(k,qcsubi,qcsubj,qcval) -> (numqcnz)
```

```
getqconk(k) -> (numqcnz,qcsubi,qcsubj,qcval)
```

Obtains all the quadratic terms in a constraint. The quadratic terms are stored sequentially in `qcsubi`, `qcsubj`, and `qcval`.

Parameters

- `k` (`int32`) – Which constraint. (input)
- `qcsubi` (`int32[]`) – Row subscripts for quadratic constraint matrix. (output)
- `qcsubj` (`int32[]`) – Column subscripts for quadratic constraint matrix. (output)
- `qcval` (`float[]`) – Quadratic constraint coefficient values. (output)

Return

- `numqcnz` (`int64`) – Number of quadratic terms.
- `qcsubi` (`int32[]`) – Row subscripts for quadratic constraint matrix.
- `qcsubj` (`int32[]`) – Column subscripts for quadratic constraint matrix.
- `qcval` (`float[]`) – Quadratic constraint coefficient values.

Groups

Inspecting the task, *Problem data - quadratic part*, *Problem data - constraints*

`Task.getqobj`

```
getqobj(qosubi,qosubj,qoval) -> (numqonz)
```

```
getqobj() -> (numqonz,qosubi,qosubj,qoval)
```

Obtains the quadratic terms in the objective. The required quadratic terms are stored sequentially in `qosubi`, `qosubj`, and `qoval`.

Parameters

- `qosubi` (`int32[]`) – Row subscripts for quadratic objective coefficients. (output)
- `qosubj` (`int32[]`) – Column subscripts for quadratic objective coefficients. (output)
- `qoval` (`float[]`) – Quadratic objective coefficient values. (output)

Return

- `numqonz` (`int64`) – Number of non-zero elements in the quadratic objective terms.
- `qosubi` (`int32[]`) – Row subscripts for quadratic objective coefficients.
- `qosubj` (`int32[]`) – Column subscripts for quadratic objective coefficients.
- `qoval` (`float[]`) – Quadratic objective coefficient values.

Groups

Inspecting the task, *Problem data - quadratic part*

`Task.getqobjij`

```
getqobjij(i,j) -> (qoij)
```

Obtains one coefficient q^o_{ij} in the quadratic term of the objective.

Parameters

- `i` (`int32`) – Row index of the coefficient. (input)
- `j` (`int32`) – Column index of coefficient. (input)

Return

`qoij` (`float`) – The required coefficient.

Groups

Inspecting the task, *Problem data - quadratic part*

`Task.getreducedcosts`

```
getreducedcosts(whichsol,first,last,redcosts)
```

```
getreducedcosts(whichsol,first,last) -> (redcosts)
```

Computes the reduced costs for a slice of variables and returns them in the array `redcosts` i.e.

$$\texttt{redcosts} = [(s_l^x)_j - (s_u^x)_j,\ j = \texttt{first}, \ldots, \texttt{last} - 1] \tag{15.2}$$

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (`int32`) – The index of the first variable in the sequence. (input)
- `last` (`int32`) – The index of the last variable in the sequence plus 1. (input)
- `redcosts` (`float[]`) – The reduced costs for the required slice of variables. (output)

Return

`redcosts` (`float[]`) – The reduced costs for the required slice of variables.

Groups

Solution - dual

`Task.getskc`

```
getskc(whichsol,skc)
```

```
getskc(whichsol) -> (skc)
```

Obtains the status keys for the constraints.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `skc` (*mosek.stakey* `[]`) – Status keys for the constraints. (output)

Return

`skc` (*mosek.stakey* `[]`) – Status keys for the constraints.

Groups

Solution information

`Task.getskcslice`

```
getskcslice(whichsol,first,last,skc)
```

```
getskcslice(whichsol,first,last) -> (skc)
```

Obtains the status keys for a slice of the constraints.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (`int32`) – First index in the sequence. (input)
- `last` (`int32`) – Last index plus 1 in the sequence. (input)
- `skc` (*mosek.stakey* `[]`) – Status keys for the constraints. (output)

Return

`skc` (*mosek.stakey* `[]`) – Status keys for the constraints.

Groups

Solution information

`Task.getskn`

```
getskn(whichsol,skn)
```

```
getskn(whichsol) -> (skn)
```

Obtains the status keys for the conic constraints.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `skn` (*mosek.stakey* `[]`) – Status keys for the conic constraints. (output)

Return

`skn` (*mosek.stakey* `[]`) – Status keys for the conic constraints.

Groups

Solution information

`Task.getskx`

```
getskx(whichsol,skx)
```

```
getskx(whichsol) -> (skx)
```

Obtains the status keys for the scalar variables.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `skx` (*mosek.stakey* `[]`) – Status keys for the variables. (output)

Return

`skx` (*mosek.stakey* `[]`) – Status keys for the variables.

Groups

Solution information

`Task.getskxslice`

```
getskxslice(whichsol,first,last,skx)
```

```
getskxslice(whichsol,first,last) -> (skx)
```

Obtains the status keys for a slice of the scalar variables.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (`int32`) – First index in the sequence. (input)
- `last` (`int32`) – Last index plus 1 in the sequence. (input)
- `skx` (*mosek.stakey* `[]`) – Status keys for the variables. (output)

Return

`skx` (*mosek.stakey* `[]`) – Status keys for the variables.

Groups

Solution information

`Task.getslc`

```
getslc(whichsol,slc)
```

```
getslc(whichsol) -> (slc)
```

Obtains the s_l^c vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `slc` (`float[]`) – Dual variables corresponding to the lower bounds on the constraints. (output)

Return

`slc` (`float[]`) – Dual variables corresponding to the lower bounds on the constraints.

Groups

Solution - dual

`Task.getslcslice`

```
getslcslice(whichsol,first,last,slc)
```

```
getslcslice(whichsol,first,last) -> (slc)
```

Obtains a slice of the s_l^c vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (`int32`) – First index in the sequence. (input)
- `last` (`int32`) – Last index plus 1 in the sequence. (input)
- `slc` (`float[]`) – Dual variables corresponding to the lower bounds on the constraints. (output)

Return

`slc` (`float[]`) – Dual variables corresponding to the lower bounds on the constraints.

Groups

Solution - dual

`Task.getslx`

```
getslx(whichsol,slx)
```

```
getslx(whichsol) -> (slx)
```

Obtains the s_l^x vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `slx` (`float[]`) – Dual variables corresponding to the lower bounds on the variables. (output)

Return

`slx` (`float[]`) – Dual variables corresponding to the lower bounds on the variables.

Groups

Solution - dual

`Task.getslxslice`

```
getslxslice(whichsol,first,last,slx)
```

```
getslxslice(whichsol,first,last) -> (slx)
```

Obtains a slice of the s_l^x vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (`int32`) – First index in the sequence. (input)
- `last` (`int32`) – Last index plus 1 in the sequence. (input)
- `slx` (`float[]`) – Dual variables corresponding to the lower bounds on the variables. (output)

Return
`slx` (`float[]`) – Dual variables corresponding to the lower bounds on the variables.

Groups
Solution - dual

`Task.getsnx`

```
getsnx(whichsol,snx)
```

```
getsnx(whichsol) -> (snx)
```

Obtains the s_n^x vector for a solution.

Parameters
- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `snx` (`float[]`) – Dual variables corresponding to the conic constraints on the variables. (output)

Return
`snx` (`float[]`) – Dual variables corresponding to the conic constraints on the variables.

Groups
Solution - dual

`Task.getsnxslice`

```
getsnxslice(whichsol,first,last,snx)
```

```
getsnxslice(whichsol,first,last) -> (snx)
```

Obtains a slice of the s_n^x vector for a solution.

Parameters
- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (`int32`) – First index in the sequence. (input)
- `last` (`int32`) – Last index plus 1 in the sequence. (input)
- `snx` (`float[]`) – Dual variables corresponding to the conic constraints on the variables. (output)

Return
`snx` (`float[]`) – Dual variables corresponding to the conic constraints on the variables.

Groups
Solution - dual

`Task.getsolsta`

```
getsolsta(whichsol) -> (solutionsta)
```

Obtains the solution status.

Parameters
`whichsol` (*mosek.soltype*) – Selects a solution. (input)

Return
`solutionsta` (*mosek.solsta*) – Solution status.

Groups
Solution information

`Task.getsolution`

```
getsolution(whichsol,
            skc,
            skx,
            skn,
            xc,
            xx,
            y,
            slc,
            suc,
            slx,
            sux,
            snx) -> (problemsta,solutionsta)
```

```
getsolution(whichsol) ->
           (problemsta,
            solutionsta,
            skc,
            skx,
            skn,
            xc,
            xx,
            y,
            slc,
            suc,
            slx,
            sux,
            snx)
```

Obtains the complete solution.

Consider the case of linear programming. The primal problem is given by

$$\begin{array}{lccccc} \text{minimize} & & & c^T x + c^f & & \\ \text{subject to} & l^c & \leq & Ax & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x. \end{array}$$

and the corresponding dual problem is

$$\begin{array}{lcll} \text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c & & \\ & +(l^x)^T s_l^x - (u^x)^T s_u^x + c^f & & \\ \text{subject to} & A^T y + s_l^x - s_u^x & = & c, \\ & -y + s_l^c - s_u^c & = & 0, \\ & s_l^c, s_u^c, s_l^x, s_u^x \geq 0. & & \end{array}$$

A conic optimization problem has the same primal variables as in the linear case. Recall that the dual of a conic optimization problem is given by:

$$\begin{array}{lcll} \text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c & & \\ & +(l^x)^T s_l^x - (u^x)^T s_u^x + c^f & & \\ \text{subject to} & A^T y + s_l^x - s_u^x + s_n^x & = & c, \\ & -y + s_l^c - s_u^c & = & 0, \\ & s_l^c, s_u^c, s_l^x, s_u^x & \geq & 0, \\ & s_n^x \in \mathcal{K}^* & & \end{array}$$

The mapping between variables and arguments to the function is as follows:

- `xx` : Corresponds to variable x (also denoted x^x).
- `xc` : Corresponds to $x^c := Ax$.
- `y` : Corresponds to variable y.

- `slc`: Corresponds to variable s_l^c.
- `suc`: Corresponds to variable s_u^c.
- `slx`: Corresponds to variable s_l^x.
- `sux`: Corresponds to variable s_u^x.
- `snx`: Corresponds to variable s_n^x.

The meaning of the values returned by this function depend on the *solution status* returned in the argument `solsta`. The most important possible values of `solsta` are:

- *solsta.optimal* : An optimal solution satisfying the optimality criteria for continuous problems is returned.
- *solsta.integer_optimal* : An optimal solution satisfying the optimality criteria for integer problems is returned.
- *solsta.prim_feas* : A solution satisfying the feasibility criteria.
- *solsta.prim_infeas_cer* : A primal certificate of infeasibility is returned.
- *solsta.dual_infeas_cer* : A dual certificate of infeasibility is returned.

In order to retrieve the primal and dual values of semidefinite variables see *Task.getbarxj* and *Task.getbarsj*.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `skc` (*mosek.stakey* `[]`) – Status keys for the constraints. (output)
- `skx` (*mosek.stakey* `[]`) – Status keys for the variables. (output)
- `skn` (*mosek.stakey* `[]`) – Status keys for the conic constraints. (output)
- `xc` (`float[]`) – Primal constraint solution. (output)
- `xx` (`float[]`) – Primal variable solution. (output)
- `y` (`float[]`) – Vector of dual variables corresponding to the constraints. (output)
- `slc` (`float[]`) – Dual variables corresponding to the lower bounds on the constraints. (output)
- `suc` (`float[]`) – Dual variables corresponding to the upper bounds on the constraints. (output)
- `slx` (`float[]`) – Dual variables corresponding to the lower bounds on the variables. (output)
- `sux` (`float[]`) – Dual variables corresponding to the upper bounds on the variables. (output)
- `snx` (`float[]`) – Dual variables corresponding to the conic constraints on the variables. (output)

Return

- `problemsta` (*mosek.prosta*) – Problem status.
- `solutionsta` (*mosek.solsta*) – Solution status.
- `skc` (*mosek.stakey* `[]`) – Status keys for the constraints.
- `skx` (*mosek.stakey* `[]`) – Status keys for the variables.
- `skn` (*mosek.stakey* `[]`) – Status keys for the conic constraints.
- `xc` (`float[]`) – Primal constraint solution.
- `xx` (`float[]`) – Primal variable solution.
- `y` (`float[]`) – Vector of dual variables corresponding to the constraints.
- `slc` (`float[]`) – Dual variables corresponding to the lower bounds on the constraints.
- `suc` (`float[]`) – Dual variables corresponding to the upper bounds on the constraints.
- `slx` (`float[]`) – Dual variables corresponding to the lower bounds on the variables.

- `sux` (`float[]`) – Dual variables corresponding to the upper bounds on the variables.
- `snx` (`float[]`) – Dual variables corresponding to the conic constraints on the variables.

Groups

Solution information, *Solution - primal*, *Solution - dual*

`Task.getsolutioninfo`

```
getsolutioninfo(whichsol) ->
                (pobj,
                 pviolcon,
                 pviolvar,
                 pviolbarvar,
                 pviolcone,
                 pviolitg,
                 dobj,
                 dviolcon,
                 dviolvar,
                 dviolbarvar,
                 dviolcone)
```

Obtains information about a solution.

Parameters

`whichsol` (*mosek.soltype*) – Selects a solution. (input)

Return

- `pobj` (`float`) – The primal objective value as computed by *Task.getprimalobj*.
- `pviolcon` (`float`) – Maximal primal violation of the solution associated with the x^c variables where the violations are computed by *Task.getpviolcon*.
- `pviolvar` (`float`) – Maximal primal violation of the solution for the x variables where the violations are computed by *Task.getpviolvar*.
- `pviolbarvar` (`float`) – Maximal primal violation of solution for the $\overline{X}$ variables where the violations are computed by *Task.getpviolbarvar*.
- `pviolcone` (`float`) – Maximal primal violation of solution for the conic constraints where the violations are computed by ~~*Task.getpviolcones*~~.
- `pviolitg` (`float`) – Maximal violation in the integer constraints. The violation for an integer variable x_j is given by $\min(x_j - \lfloor x_j \rfloor, \lceil x_j \rceil - x_j)$. This number is always zero for the interior-point and basic solutions.
- `dobj` (`float`) – Dual objective value as computed by *Task.getdualobj*.
- `dviolcon` (`float`) – Maximal violation of the dual solution associated with the x^c variable as computed by *Task.getdviolcon*.
- `dviolvar` (`float`) – Maximal violation of the dual solution associated with the x variable as computed by *Task.getdviolvar*.
- `dviolbarvar` (`float`) – Maximal violation of the dual solution associated with the $\overline{S}$ variable as computed by *Task.getdviolbarvar*.
- `dviolcone` (`float`) – Maximal violation of the dual solution associated with the dual conic constraints as computed by ~~*Task.getdviolcones*~~.

Groups

Solution information

`Task.getsolutioninfonew`

```
getsolutioninfonew(whichsol) ->
                   (pobj,
                    pviolcon,
```

(continues on next page)

(continued from previous page)

```
                    pviolvar,
                    pviolbarvar,
                    pviolcone,
                    pviolacc,
                    pvioldjc,
                    pviolitg,
                    dobj,
                    dviolcon,
                    dviolvar,
                    dviolbarvar,
                    dviolcone,
                    dviolacc)
```

Obtains information about a solution.

Parameters

`whichsol` (*mosek.soltype*) – Selects a solution. (input)

Return

- `pobj` (`float`) – The primal objective value as computed by *Task.getprimalobj*.
- `pviolcon` (`float`) – Maximal primal violation of the solution associated with the x^c variables where the violations are computed by *Task.getpviolcon*.
- `pviolvar` (`float`) – Maximal primal violation of the solution for the x variables where the violations are computed by *Task.getpviolvar*.
- `pviolbarvar` (`float`) – Maximal primal violation of solution for the $\overline{X}$ variables where the violations are computed by *Task.getpviolbarvar*.
- `pviolcone` (`float`) – Maximal primal violation of solution for the conic constraints where the violations are computed by ~~*Task.getpviolcones*~~.
- `pviolacc` (`float`) – Maximal primal violation of solution for the affine conic constraints where the violations are computed by *Task.getpviolacc*.
- `pvioldjc` (`float`) – Maximal primal violation of solution for the disjunctive constraints where the violations are computed by *Task.getpvioldjc*.
- `pviolitg` (`float`) – Maximal violation in the integer constraints. The violation for an integer variable x_j is given by $\min(x_j - \lfloor x_j \rfloor, \lceil x_j \rceil - x_j)$. This number is always zero for the interior-point and basic solutions.
- `dobj` (`float`) – Dual objective value as computed by *Task.getdualobj*.
- `dviolcon` (`float`) – Maximal violation of the dual solution associated with the x^c variable as computed by *Task.getdviolcon*.
- `dviolvar` (`float`) – Maximal violation of the dual solution associated with the x variable as computed by *Task.getdviolvar*.
- `dviolbarvar` (`float`) – Maximal violation of the dual solution associated with the $\overline{S}$ variable as computed by *Task.getdviolbarvar*.
- `dviolcone` (`float`) – Maximal violation of the dual solution associated with the dual conic constraints as computed by ~~*Task.getdviolcones*~~.
- `dviolacc` (`float`) – Maximal violation of the dual solution associated with the affine conic constraints as computed by *Task.getdviolacc*.

Groups

Solution information

`Task.getsolutionnew`

```
getsolutionnew(whichsol,
               skc,
               skx,
               skn,
```

(continues on next page)

(continued from previous page)

```
                xc,
                xx,
                y,
                slc,
                suc,
                slx,
                sux,
                snx,
                doty) -> (problemsta,solutionsta)
```

```
getsolutionnew(whichsol) ->
               (problemsta,
                solutionsta,
                skc,
                skx,
                skn,
                xc,
                xx,
                y,
                slc,
                suc,
                slx,
                sux,
                snx,
                doty)
```

Obtains the complete solution. See *Task.getsolution* for further information.

In order to retrieve the primal and dual values of semidefinite variables see *Task.getbarxj* and *Task.getbarsj*.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `skc` (*mosek.stakey*`[]`) – Status keys for the constraints. (output)
- `skx` (*mosek.stakey*`[]`) – Status keys for the variables. (output)
- `skn` (*mosek.stakey*`[]`) – Status keys for the conic constraints. (output)
- `xc` (`float[]`) – Primal constraint solution. (output)
- `xx` (`float[]`) – Primal variable solution. (output)
- `y` (`float[]`) – Vector of dual variables corresponding to the constraints. (output)
- `slc` (`float[]`) – Dual variables corresponding to the lower bounds on the constraints. (output)
- `suc` (`float[]`) – Dual variables corresponding to the upper bounds on the constraints. (output)
- `slx` (`float[]`) – Dual variables corresponding to the lower bounds on the variables. (output)
- `sux` (`float[]`) – Dual variables corresponding to the upper bounds on the variables. (output)
- `snx` (`float[]`) – Dual variables corresponding to the conic constraints on the variables. (output)
- `doty` (`float[]`) – Dual variables corresponding to affine conic constraints. (output)

Return

- `problemsta` (*mosek.prosta*) – Problem status.
- `solutionsta` (*mosek.solsta*) – Solution status.
- `skc` (*mosek.stakey*`[]`) – Status keys for the constraints.

- `skx` (*mosek.stakey* `[]`) – Status keys for the variables.
- `skn` (*mosek.stakey* `[]`) – Status keys for the conic constraints.
- `xc` (`float[]`) – Primal constraint solution.
- `xx` (`float[]`) – Primal variable solution.
- `y` (`float[]`) – Vector of dual variables corresponding to the constraints.
- `slc` (`float[]`) – Dual variables corresponding to the lower bounds on the constraints.
- `suc` (`float[]`) – Dual variables corresponding to the upper bounds on the constraints.
- `slx` (`float[]`) – Dual variables corresponding to the lower bounds on the variables.
- `sux` (`float[]`) – Dual variables corresponding to the upper bounds on the variables.
- `snx` (`float[]`) – Dual variables corresponding to the conic constraints on the variables.
- `doty` (`float[]`) – Dual variables corresponding to affine conic constraints.

Groups

Solution information, *Solution - primal*, *Solution - dual*

`Task.getsolutionslice`

```
getsolutionslice(whichsol,solitem,first,last,values)
```

```
getsolutionslice(whichsol,solitem,first,last) -> (values)
```

Obtains a slice of one item from the solution. The format of the solution is exactly as in *Task.getsolution*. The parameter `solitem` determines which of the solution vectors should be returned.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `solitem` (*mosek.solitem*) – Which part of the solution is required. (input)
- `first` (`int32`) – First index in the sequence. (input)
- `last` (`int32`) – Last index plus 1 in the sequence. (input)
- `values` (`float[]`) – The values in the required sequence are stored sequentially in `values`. (output)

Return

`values` (`float[]`) – The values in the required sequence are stored sequentially in `values`.

Groups

Solution - primal, *Solution - dual*, *Solution information*

`Task.getsparsesymmat`

```
getsparsesymmat(idx,subi,subj,valij)
```

```
getsparsesymmat(idx) -> (subi,subj,valij)
```

Get a single symmetric matrix from the matrix store.

Parameters

- `idx` (`int64`) – Index of the matrix to retrieve. (input)
- `subi` (`int32[]`) – Row subscripts of the matrix non-zero elements. (output)
- `subj` (`int32[]`) – Column subscripts of the matrix non-zero elements. (output)
- `valij` (`float[]`) – Coefficients of the matrix non-zero elements. (output)

Return

- `subi` (`int32[]`) – Row subscripts of the matrix non-zero elements.

- `subj` (`int32[]`) – Column subscripts of the matrix non-zero elements.
- `valij` (`float[]`) – Coefficients of the matrix non-zero elements.

Groups
Problem data - semidefinite, *Inspecting the task*

`Task.getstrparam`

```
getstrparam(param) -> (len,parvalue)
```

Obtains the value of a string parameter.

Parameters
`param` (*mosek.sparam*) – Which parameter. (input)

Return
- `len` (`int32`) – The length of the parameter value.
- `parvalue` (`str`) – Parameter value.

Groups
Names, *Parameters*

`Task.getstrparamlen`

```
getstrparamlen(param) -> (len)
```

Obtains the length of a string parameter.

Parameters
`param` (*mosek.sparam*) – Which parameter. (input)

Return
`len` (`int32`) – The length of the parameter value.

Groups
Names, *Parameters*

`Task.getsuc`

```
getsuc(whichsol,suc)
```

```
getsuc(whichsol) -> (suc)
```

Obtains the s_u^c vector for a solution.

Parameters
- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `suc` (`float[]`) – Dual variables corresponding to the upper bounds on the constraints. (output)

Return
`suc` (`float[]`) – Dual variables corresponding to the upper bounds on the constraints.

Groups
Solution - dual

`Task.getsucslice`

```
getsucslice(whichsol,first,last,suc)
```

```
getsucslice(whichsol,first,last) -> (suc)
```

Obtains a slice of the s_u^c vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (`int32`) – First index in the sequence. (input)
- `last` (`int32`) – Last index plus 1 in the sequence. (input)
- `suc` (`float[]`) – Dual variables corresponding to the upper bounds on the constraints. (output)

Return

`suc` (`float[]`) – Dual variables corresponding to the upper bounds on the constraints.

Groups

Solution - dual

`Task.getsux`

```
getsux(whichsol,sux)
```

```
getsux(whichsol) -> (sux)
```

Obtains the s_u^x vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `sux` (`float[]`) – Dual variables corresponding to the upper bounds on the variables. (output)

Return

`sux` (`float[]`) – Dual variables corresponding to the upper bounds on the variables.

Groups

Solution - dual

`Task.getsuxslice`

```
getsuxslice(whichsol,first,last,sux)
```

```
getsuxslice(whichsol,first,last) -> (sux)
```

Obtains a slice of the s_u^x vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (`int32`) – First index in the sequence. (input)
- `last` (`int32`) – Last index plus 1 in the sequence. (input)
- `sux` (`float[]`) – Dual variables corresponding to the upper bounds on the variables. (output)

Return

`sux` (`float[]`) – Dual variables corresponding to the upper bounds on the variables.

Groups

Solution - dual

`Task.getsymmatinfo`

```
getsymmatinfo(idx) -> (dim,nz,mattype)
```

MOSEK maintains a vector denoted by E of symmetric data matrices. This function makes it possible to obtain important information about a single matrix in E.

Parameters

`idx` (`int64`) – Index of the matrix for which information is requested. (input)

Return

- `dim` (`int32`) – Returns the dimension of the requested matrix.
- `nz` (`int64`) – Returns the number of non-zeros in the requested matrix.
- `mattype` (*mosek.symmattype*) – Returns the type of the requested matrix.

Groups
Problem data - semidefinite, *Inspecting the task*

`Task.gettaskname`

```
gettaskname() -> (taskname)
```

Obtains the name assigned to the task.

Return
`taskname` (`str`) – Returns the task name.

Groups
Names, *Inspecting the task*

`Task.gettasknamelen`

```
gettasknamelen() -> (len)
```

Obtains the length the task name.

Return
`len` (`int32`) – Returns the length of the task name.

Groups
Names, *Inspecting the task*

`Task.getvarbound`

```
getvarbound(i) -> (bk,bl,bu)
```

Obtains bound information for one variable.

Parameters
`i` (`int32`) – Index of the variable for which the bound information should be obtained. (input)

Return
- `bk` (*mosek.boundkey*) – Bound keys.
- `bl` (`float`) – Values for lower bounds.
- `bu` (`float`) – Values for upper bounds.

Groups
Problem data - linear part, *Inspecting the task*, *Problem data - bounds*, *Problem data - variables*

`Task.getvarboundslice`

```
getvarboundslice(first,last,bk,bl,bu)
```

```
getvarboundslice(first,last) -> (bk,bl,bu)
```

Obtains bounds information for a slice of the variables.

Parameters
- `first` (`int32`) – First index in the sequence. (input)
- `last` (`int32`) – Last index plus 1 in the sequence. (input)
- `bk` (*mosek.boundkey*`[]`) – Bound keys. (output)
- `bl` (`float[]`) – Values for lower bounds. (output)
- `bu` (`float[]`) – Values for upper bounds. (output)

Return

- `bk` (*mosek.boundkey* `[]`) – Bound keys.
- `bl` (`float[]`) – Values for lower bounds.
- `bu` (`float[]`) – Values for upper bounds.

Groups

Problem data - linear part, *Inspecting the task*, *Problem data - bounds*, *Problem data - variables*

`Task.getvarname`

```
getvarname(j) -> (name)
```

Obtains the name of a variable.

Parameters

`j` (`int32`) – Index of a variable. (input)

Return

`name` (`str`) – Returns the required name.

Groups

Names, *Problem data - linear part*, *Problem data - variables*, *Inspecting the task*

`Task.getvarnameindex`

```
getvarnameindex(somename) -> (asgn,index)
```

Checks whether the name `somename` has been assigned to any variable. If so, the index of the variable is reported.

Parameters

`somename` (`str`) – The name which should be checked. (input)

Return

- `asgn` (`int32`) – Is non-zero if the name `somename` is assigned to a variable.
- `index` (`int32`) – If the name `somename` is assigned to a variable, then `index` is the index of the variable.

Groups

Names, *Problem data - linear part*, *Problem data - variables*, *Inspecting the task*

`Task.getvarnamelen`

```
getvarnamelen(i) -> (len)
```

Obtains the length of the name of a variable.

Parameters

`i` (`int32`) – Index of a variable. (input)

Return

`len` (`int32`) – Returns the length of the indicated name.

Groups

Names, *Problem data - linear part*, *Problem data - variables*, *Inspecting the task*

`Task.getvartype`

```
getvartype(j) -> (vartype)
```

Gets the variable type of one variable.

Parameters

`j` (`int32`) – Index of the variable. (input)

Return

`vartype` (*mosek.variabletype*) – Variable type of the j-th variable.

Groups
Inspecting the task, *Problem data - variables*

`Task.getvartypelist`

```
getvartypelist(subj,vartype)
```

```
getvartypelist(subj) -> (vartype)
```

Obtains the variable type of one or more variables. Upon return `vartype[k]` is the variable type of variable `subj[k]`.

Parameters
- `subj` (`int32[]`) – A list of variable indexes. (input)
- `vartype` (*`mosek.variabletype`* `[]`) – The variables types corresponding to the variables specified by `subj`. (output)

Return
`vartype` (*`mosek.variabletype`* `[]`) – The variables types corresponding to the variables specified by `subj`.

Groups
Inspecting the task, *Problem data - variables*

`Task.getxc`

```
getxc(whichsol,xc)
```

```
getxc(whichsol) -> (xc)
```

Obtains the x^c vector for a solution.

Parameters
- `whichsol` (*`mosek.soltype`*) – Selects a solution. (input)
- `xc` (`float[]`) – Primal constraint solution. (output)

Return
`xc` (`float[]`) – Primal constraint solution.

Groups
Solution - primal

`Task.getxcslice`

```
getxcslice(whichsol,first,last,xc)
```

```
getxcslice(whichsol,first,last) -> (xc)
```

Obtains a slice of the x^c vector for a solution.

Parameters
- `whichsol` (*`mosek.soltype`*) – Selects a solution. (input)
- `first` (`int32`) – First index in the sequence. (input)
- `last` (`int32`) – Last index plus 1 in the sequence. (input)
- `xc` (`float[]`) – Primal constraint solution. (output)

Return
`xc` (`float[]`) – Primal constraint solution.

Groups
Solution - primal

`Task.getxx`

```
getxx(whichsol,xx)
```

```
getxx(whichsol) -> (xx)
```

Obtains the x^x vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `xx` (`float[]`) – Primal variable solution. (output)

Return

`xx` (`float[]`) – Primal variable solution.

Groups

Solution - primal

`Task.getxxslice`

```
getxxslice(whichsol,first,last,xx)
```

```
getxxslice(whichsol,first,last) -> (xx)
```

Obtains a slice of the x^x vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (`int32`) – First index in the sequence. (input)
- `last` (`int32`) – Last index plus 1 in the sequence. (input)
- `xx` (`float[]`) – Primal variable solution. (output)

Return

`xx` (`float[]`) – Primal variable solution.

Groups

Solution - primal

`Task.gety`

```
gety(whichsol,y)
```

```
gety(whichsol) -> (y)
```

Obtains the y vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `y` (`float[]`) – Vector of dual variables corresponding to the constraints. (output)

Return

`y` (`float[]`) – Vector of dual variables corresponding to the constraints.

Groups

Solution - dual

`Task.getyslice`

```
getyslice(whichsol,first,last,y)
```

```
getyslice(whichsol,first,last) -> (y)
```

Obtains a slice of the y vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (`int32`) – First index in the sequence. (input)
- `last` (`int32`) – Last index plus 1 in the sequence. (input)
- `y` (`float[]`) – Vector of dual variables corresponding to the constraints. (output)

Return

`y` (`float[]`) – Vector of dual variables corresponding to the constraints.

Groups

Solution - dual

`Task.infeasibilityreport`

```
infeasibilityreport(whichstream,whichsol)
```

Prints the infeasibility report to an output stream.

Parameters

- `whichstream` (*mosek.streamtype*) – Index of the stream. (input)
- `whichsol` (*mosek.soltype*) – Selects a solution. (input)

Groups

Infeasibility diagnostic

`Task.initbasissolve`

```
initbasissolve(basis)
```

```
initbasissolve() -> (basis)
```

Prepare a task for use with the *Task.solvewithbasis* function.

This function should be called

- immediately before the first call to *Task.solvewithbasis*, and
- immediately before any subsequent call to *Task.solvewithbasis* if the task has been modified.

If the basis is singular i.e. not invertible, then the error *rescode.err_basis_singular* is reported.

Parameters

`basis` (`int32[]`) – The array of basis indexes to use. The array is interpreted as follows: If $\texttt{basis}[i] \leq numcon-1$, then $x^c_{\texttt{basis}[i]}$ is in the basis at position i, otherwise $x_{\texttt{basis}[i]-\texttt{numcon}}$ is in the basis at position i. (output)

Return

`basis` (`int32[]`) – The array of basis indexes to use. The array is interpreted as follows: If $\texttt{basis}[i] \leq numcon-1$, then $x^c_{\texttt{basis}[i]}$ is in the basis at position i, otherwise $x_{\texttt{basis}[i]-\texttt{numcon}}$ is in the basis at position i.

Groups

Solving systems with basis matrix

`Task.inputdata`

```
inputdata(maxnumcon,
          maxnumvar,
          c,
          cfix,
          aptrb,
          aptre,
          asub,
          aval,
```

(continues on next page)

(continued from previous page)

```
          bkc,
          blc,
          buc,
          bkx,
          blx,
          bux)
```

Input the linear part of an optimization problem.

The non-zeros of A are inputted column-wise in the format described in Section *Column or Row Ordered Sparse Matrix*.

For an explained code example see Section *Linear Optimization* and Section *Matrix Formats*.

Parameters

- `maxnumcon` (`int32`) – Number of preallocated constraints in the optimization task. (input)
- `maxnumvar` (`int32`) – Number of preallocated variables in the optimization task. (input)
- `c` (`float[]`) – Linear terms of the objective as a dense vector. The length is the number of variables. (input)
- `cfix` (`float`) – Fixed term in the objective. (input)
- `aptrb` (`int64[]`) – Row or column start pointers. (input)
- `aptre` (`int64[]`) – Row or column end pointers. (input)
- `asub` (`int32[]`) – Coefficient subscripts. (input)
- `aval` (`float[]`) – Coefficient values. (input)
- `bkc` (*`mosek.boundkey`* `[]`) – Bound keys for the constraints. (input)
- `blc` (`float[]`) – Lower bounds for the constraints. (input)
- `buc` (`float[]`) – Upper bounds for the constraints. (input)
- `bkx` (*`mosek.boundkey`* `[]`) – Bound keys for the variables. (input)
- `blx` (`float[]`) – Lower bounds for the variables. (input)
- `bux` (`float[]`) – Upper bounds for the variables. (input)

Groups

Problem data - linear part, *Problem data - bounds*, *Problem data - constraints*

`Task.isdouparname`

```
isdouparname(parname) -> (param)
```

Checks whether `parname` is a valid double parameter name.

Parameters

`parname` (`str`) – Parameter name. (input)

Return

`param` (*`mosek.dparam`*) – Returns the parameter corresponding to the name, if one exists.

Groups

Parameters, *Names*

`Task.isintparname`

```
isintparname(parname) -> (param)
```

Checks whether `parname` is a valid integer parameter name.

Parameters

`parname` (`str`) – Parameter name. (input)

Return
`param` (*mosek.iparam*) – Returns the parameter corresponding to the name, if one exists.

Groups
Parameters, *Names*

`Task.isstrparname`

```
isstrparname(parname) -> (param)
```

Checks whether `parname` is a valid string parameter name.

Parameters
`parname` (`str`) – Parameter name. (input)

Return
`param` (*mosek.sparam*) – Returns the parameter corresponding to the name, if one exists.

Groups
Parameters, *Names*

`Task.linkfiletostream`

```
linkfiletostream(whichstream,filename,append)
```

Directs all output from a task stream `whichstream` to a file `filename`.

Parameters
- `whichstream` (*mosek.streamtype*) – Index of the stream. (input)
- `filename` (`str`) – A valid file name. (input)
- `append` (`int32`) – If this argument is 0 the output file will be overwritten, otherwise it will be appended to. (input)

Groups
Logging

`Task.onesolutionsummary`

```
onesolutionsummary(whichstream,whichsol)
```

Prints a short summary of a specified solution.

Parameters
- `whichstream` (*mosek.streamtype*) – Index of the stream. (input)
- `whichsol` (*mosek.soltype*) – Selects a solution. (input)

Groups
Logging, *Solution information*

`Task.optimize`

```
optimize() -> (trmcode)
```

Calls the optimizer. Depending on the problem type and the selected optimizer this will call one of the optimizers in **MOSEK**. By default the interior point optimizer will be selected for continuous problems. The optimizer may be selected manually by setting the parameter *iparam.optimizer*.

Return
`trmcode` (*mosek.rescode*) – Is either *rescode.ok* or a termination response code.

Groups
Optimization

`Task.optimizermt`

```
optimizermt(address,accesstoken) -> (trmcode)
```

Offload the optimization task to an instance of OptServer specified by `addr`, which should be a valid URL, for example `http://server:port` or `https://server:port`. The call will block until a result is available or the connection closes.

If the server requires authentication, the authentication token can be passed in the `accesstoken` argument.

If the server requires encryption, the keys can be passed using one of the solver parameters *sparam.remote_tls_cert* or *sparam.remote_tls_cert_path*.

Parameters

- `address` (`str`) – Address of the OptServer. (input)
- `accesstoken` (`str`) – Access token. (input)

Return

`trmcode` (*mosek.rescode*) – Is either *rescode.ok* or a termination response code.

Groups

Remote optimization

`Task.optimizersummary`

```
optimizersummary(whichstream)
```

Prints a short summary with optimizer statistics from last optimization.

Parameters

`whichstream` (*mosek.streamtype*) – Index of the stream. (input)

Groups

Logging

`Task.primalrepair`

```
primalrepair(wlc,wuc,wlx,wux)
```

The function repairs a primal infeasible optimization problem by adjusting the bounds on the constraints and variables where the adjustment is computed as the minimal weighted sum of relaxations to the bounds on the constraints and variables. Observe the function only repairs the problem but does not solve it. If an optimal solution is required the problem should be optimized after the repair.

The function is applicable to linear and conic problems possibly with integer variables.

Observe that when computing the minimal weighted relaxation the termination tolerance specified by the parameters of the task is employed. For instance the parameter *iparam.mio_mode* can be used to make **MOSEK** ignore the integer constraints during the repair which usually leads to a much faster repair. However, the drawback is of course that the repaired problem may not have an integer feasible solution.

Note the function modifies the task in place. If this is not desired, then apply the function to a cloned task.

Parameters

- `wlc` (`float[]`) – $(w_l^c)_i$ is the weight associated with relaxing the lower bound on constraint i. If the weight is negative, then the lower bound is not relaxed. Moreover, if the argument is `None`, then all the weights are assumed to be 1. (input)
- `wuc` (`float[]`) – $(w_u^c)_i$ is the weight associated with relaxing the upper bound on constraint i. If the weight is negative, then the upper bound is not relaxed. Moreover, if the argument is `None`, then all the weights are assumed to be 1. (input)

- `wlx` (`float[]`) – $(w_l^x)_j$ is the weight associated with relaxing the lower bound on variable j. If the weight is negative, then the lower bound is not relaxed. Moreover, if the argument is `None`, then all the weights are assumed to be 1. (input)
- `wux` (`float[]`) – $(w_l^x)_i$ is the weight associated with relaxing the upper bound on variable j. If the weight is negative, then the upper bound is not relaxed. Moreover, if the argument is `None`, then all the weights are assumed to be 1. (input)

Groups

Infeasibility diagnostic

`Task.primalsensitivity`

```
primalsensitivity(subi,
                  marki,
                  subj,
                  markj,
                  leftpricei,
                  rightpricei,
                  leftrangei,
                  rightrangei,
                  leftpricej,
                  rightpricej,
                  leftrangej,
                  rightrangej)
```

```
primalsensitivity(subi,marki,subj,markj) ->
                 (leftpricei,
                  rightpricei,
                  leftrangei,
                  rightrangei,
                  leftpricej,
                  rightpricej,
                  leftrangej,
                  rightrangej)
```

Calculates sensitivity information for bounds on variables and constraints. For details on sensitivity analysis, the definitions of *shadow price* and *linearity interval* and an example see Section *Sensitivity Analysis*.

The type of sensitivity analysis to be performed (basis or optimal partition) is controlled by the parameter *iparam.sensitivity_type*.

Parameters

- `subi` (`int32[]`) – Indexes of constraints to analyze. (input)
- `marki` (*mosek.mark*`[]`) – The value of `marki[i]` indicates for which bound of constraint `subi[i]` sensitivity analysis is performed. If `marki[i]` = *mark.up* the upper bound of constraint `subi[i]` is analyzed, and if `marki[i]` = *mark.lo* the lower bound is analyzed. If `subi[i]` is an equality constraint, either *mark.lo* or *mark.up* can be used to select the constraint for sensitivity analysis. (input)
- `subj` (`int32[]`) – Indexes of variables to analyze. (input)
- `markj` (*mosek.mark*`[]`) – The value of `markj[j]` indicates for which bound of variable `subj[j]` sensitivity analysis is performed. If `markj[j]` = *mark.up* the upper bound of variable `subj[j]` is analyzed, and if `markj[j]` = *mark.lo* the lower bound is analyzed. If `subj[j]` is a fixed variable, either *mark.lo* or *mark.up* can be used to select the bound for sensitivity analysis. (input)
- `leftpricei` (`float[]`) – `leftpricei[i]` is the left shadow price for the bound `marki[i]` of constraint `subi[i]`. (output)

- `rightpricei` (`float[]`) – `rightpricei[i]` is the right shadow price for the bound `marki[i]` of constraint `subi[i]`. (output)
- `leftrangei` (`float[]`) – `leftrangei[i]` is the left range β_1 for the bound `marki[i]` of constraint `subi[i]`. (output)
- `rightrangei` (`float[]`) – `rightrangei[i]` is the right range β_2 for the bound `marki[i]` of constraint `subi[i]`. (output)
- `leftpricej` (`float[]`) – `leftpricej[j]` is the left shadow price for the bound `markj[j]` of variable `subj[j]`. (output)
- `rightpricej` (`float[]`) – `rightpricej[j]` is the right shadow price for the bound `markj[j]` of variable `subj[j]`. (output)
- `leftrangej` (`float[]`) – `leftrangej[j]` is the left range β_1 for the bound `markj[j]` of variable `subj[j]`. (output)
- `rightrangej` (`float[]`) – `rightrangej[j]` is the right range β_2 for the bound `markj[j]` of variable `subj[j]`. (output)

Return

- `leftpricei` (`float[]`) – `leftpricei[i]` is the left shadow price for the bound `marki[i]` of constraint `subi[i]`.
- `rightpricei` (`float[]`) – `rightpricei[i]` is the right shadow price for the bound `marki[i]` of constraint `subi[i]`.
- `leftrangei` (`float[]`) – `leftrangei[i]` is the left range β_1 for the bound `marki[i]` of constraint `subi[i]`.
- `rightrangei` (`float[]`) – `rightrangei[i]` is the right range β_2 for the bound `marki[i]` of constraint `subi[i]`.
- `leftpricej` (`float[]`) – `leftpricej[j]` is the left shadow price for the bound `markj[j]` of variable `subj[j]`.
- `rightpricej` (`float[]`) – `rightpricej[j]` is the right shadow price for the bound `markj[j]` of variable `subj[j]`.
- `leftrangej` (`float[]`) – `leftrangej[j]` is the left range β_1 for the bound `markj[j]` of variable `subj[j]`.
- `rightrangej` (`float[]`) – `rightrangej[j]` is the right range β_2 for the bound `markj[j]` of variable `subj[j]`.

Groups

Sensitivity analysis

`Task.putacc`

```
putacc(accidx,domidx,afeidxlist,b)
```

Puts an affine conic constraint. This method overwrites an existing affine conic constraint number `accidx` with new data specified in the same format as in *`Task.appendacc`*.

Parameters

- `accidx` (`int64`) – Affine conic constraint index. (input)
- `domidx` (`int64`) – Domain index. (input)
- `afeidxlist` (`int64[]`) – List of affine expression indexes. (input)
- `b` (`float[]`) – The vector of constant terms modifying affine expressions. Optional, can be `None` if not required. (input)

Groups

Problem data - affine conic constraints

`Task.putaccb`

```
putaccb(accidx,b)
```

Updates an existing affine conic constraint number `accidx` by putting a new vector b.

Parameters

- `accidx` (`int64`) – Affine conic constraint index. (input)
- `b` (`float[]`) – The vector of constant terms modifying affine expressions. Optional, can be `None` if not required. (input)

Groups

Problem data - affine conic constraints

`Task.putaccbj`

```
putaccbj(accidx,j,bj)
```

Sets one value $b[j]$ in the b vector for the affine conic constraint number `accidx`.

Parameters

- `accidx` (`int64`) – Affine conic constraint index. (input)
- `j` (`int64`) – The index of an element in b to change. (input)
- `bj` (`float`) – The new value of $b[j]$. (input)

Groups

Problem data - affine conic constraints

`Task.putaccdoty`

```
putaccdoty(whichsol,accidx,doty)
```

```
putaccdoty(whichsol,accidx) -> (doty)
```

Puts the $\dot{y}$ vector for a solution (the dual values of an affine conic constraint).

Parameters

- `whichsol` (*`mosek.soltype`*) – Selects a solution. (input)
- `accidx` (`int64`) – The index of the affine conic constraint. (input)
- `doty` (`float[]`) – The dual values for this affine conic constraint. The array should have length equal to the dimension of the constraint. (output)

Return

`doty` (`float[]`) – The dual values for this affine conic constraint. The array should have length equal to the dimension of the constraint.

Groups

Solution - dual, *Problem data - affine conic constraints*

`Task.putacclist`

```
putacclist(accidxs,domidxs,afeidxlist,b)
```

Puts affine conic constraints. This method overwrites existing affine conic constraints whose numbers are provided in the list `accidxs` with new data which is a concatenation of individual constraint descriptions in the same format as in *`Task.appendacc`* (see also *`Task.appendaccs`*).

Parameters

- `accidxs` (`int64[]`) – Affine conic constraint indices. (input)
- `domidxs` (`int64[]`) – Domain indices. (input)
- `afeidxlist` (`int64[]`) – List of affine expression indexes. (input)
- `b` (`float[]`) – The vector of constant terms modifying affine expressions. Optional, can be `None` if not required. (input)

Groups

Problem data - affine conic constraints

`Task.putaccname`

```
putaccname(accidx,name)
```

Sets the name of an affine conic constraint.

Parameters

- `accidx` (`int64`) – Index of the affine conic constraint. (input)
- `name` (`str`) – The name of the affine conic constraint. (input)

Groups

Names, *Problem data - affine conic constraints*

`Task.putacol`

```
putacol(j,subj,valj)
```

Change one column of the linear constraint matrix A. Resets all the elements in column j to zero and then sets

$$a_{\texttt{subj}[k],\texttt{j}} = \texttt{valj}[k], \quad k = 0, \ldots, \texttt{nzj} - 1.$$

Parameters

- `j` (`int32`) – Index of a column in A. (input)
- `subj` (`int32[]`) – Row indexes of non-zero values in column j of A. (input)
- `valj` (`float[]`) – New non-zero values of column j in A. (input)

Groups

Problem data - linear part

`Task.putacollist`

```
putacollist(sub,ptrb,ptre,asub,aval)
```

Change a set of columns in the linear constraint matrix A with data in sparse triplet format. The requested columns are set to zero and then updated with:

$$\begin{array}{ll} \texttt{for} & i = 0, \ldots, \texttt{num} - 1 \\ & a_{\texttt{asub}[k],\texttt{sub}[i]} = \texttt{aval}[k], \quad k = \texttt{ptrb}[i], \ldots, \texttt{ptre}[i] - 1. \end{array}$$

Parameters

- `sub` (`int32[]`) – Indexes of columns that should be replaced, no duplicates. (input)
- `ptrb` (`int64[]`) – Array of pointers to the first element in each column. (input)
- `ptre` (`int64[]`) – Array of pointers to the last element plus one in each column. (input)
- `asub` (`int32[]`) – Row indexes of new elements. (input)
- `aval` (`float[]`) – Coefficient values. (input)

Groups

Problem data - linear part

`Task.putacolslice`

```
putacolslice(first,last,ptrb,ptre,asub,aval)
```

Change a slice of columns in the linear constraint matrix A with data in sparse triplet format. The requested columns are set to zero and then updated with:

$$\begin{array}{ll} \texttt{for} & i = \texttt{first}, \ldots, \texttt{last} - 1 \\ & a_{\texttt{asub}[k],i} = \texttt{aval}[k], \quad k = \texttt{ptrb}[i - \texttt{first}], \ldots, \texttt{ptre}[i - \texttt{first}] - 1. \end{array}$$

Parameters

- `first` (`int32`) – First column in the slice. (input)

- `last` (`int32`) – Last column plus one in the slice. (input)
- `ptrb` (`int64[]`) – Array of pointers to the first element in each column. (input)
- `ptre` (`int64[]`) – Array of pointers to the last element plus one in each column. (input)
- `asub` (`int32[]`) – Row indexes of new elements. (input)
- `aval` (`float[]`) – Coefficient values. (input)

Groups

Problem data - linear part

`Task.putafebarfblocktriplet`

```
putafebarfblocktriplet(afeidx,barvaridx,subk,subl,valkl)
```

Inputs the $\overline{F}$ matrix data in block triplet form.

Parameters

- `afeidx` (`int64[]`) – Constraint index. (input)
- `barvaridx` (`int32[]`) – Symmetric matrix variable index. (input)
- `subk` (`int32[]`) – Block row index. (input)
- `subl` (`int32[]`) – Block column index. (input)
- `valkl` (`float[]`) – The numerical value associated with each block triplet. (input)

Groups

Problem data - affine expressions, *Problem data - semidefinite*

`Task.putafebarfentry`

```
putafebarfentry(afeidx,barvaridx,termidx,termweight)
```

This function sets one entry $\overline{F}_{ij}$ where $i =$ afeidx is the row index in the store of affine expressions and $j =$ barvaridx is the index of a symmetric variable. That is, the expression

$$\langle \overline{F}_{ij}, \overline{X}_j \rangle$$

will be added to the i-th affine expression.

The matrix $\overline{F}_{ij}$ is specified as a weighted sum of symmetric matrices from the symmetric matrix storage E, so $\overline{F}_{ij}$ is a symmetric matrix, precisely:

$$\overline{F}_{\text{afeidx,barvaridx}} = \sum_k \text{termweight}[k] \cdot E_{\text{termidx}[k]}.$$

By default all elements in $\overline{F}$ are 0, so only non-zero elements need be added. Setting the same entry again will overwrite the earlier entry.

The symmetric matrices from E are defined separately using the function *Task.appendsparsesymmat*.

Parameters

- `afeidx` (`int64`) – Row index of $\overline{F}$. (input)
- `barvaridx` (`int32`) – Semidefinite variable index. (input)
- `termidx` (`int64[]`) – Indices in E of the matrices appearing in the weighted sum for the $\overline{F}$ entry being specified. (input)
- `termweight` (`float[]`) – `termweight[k]` is the coefficient of the `termidx[k]`-th element of E in the weighted sum the $\overline{F}$ entry being specified. (input)

Groups

Problem data - affine expressions, *Problem data - semidefinite*

`Task.putafebarfentrylist`

```
putafebarfentrylist(afeidx,
                    barvaridx,
                    numterm,
                    ptrterm,
                    termidx,
                    termweight)
```

This function sets a list of entries in $\overline{F}$. Each entry should be described as in *Task.putafebarfentry* and all those descriptions should be combined (for example concatenated) in the input to this method. That means the k-th entry set will have row index `afeidx[k]`, symmetric variable index `barvaridx[k]` and the description of this term consists of indices in E and weights appearing in positions

$$\text{ptrterm}[k], \ldots, \text{ptrterm}[k] + (\text{lenterm}[k] - 1)$$

in the corresponding arrays `termidx` and `termweight`. See *Task.putafebarfentry* for details.

Parameters

- `afeidx` (`int64[]`) – Row indexes of $\overline{F}$. (input)
- `barvaridx` (`int32[]`) – Semidefinite variable indexes. (input)
- `numterm` (`int64[]`) – The number of terms in the weighted sums that form each entry. (input)
- `ptrterm` (`int64[]`) – The pointer to the beginning of the description of each entry. (input)
- `termidx` (`int64[]`) – Concatenated lists of indices in E of the matrices appearing in the weighted sums for the $\overline{F}$ being specified. (input)
- `termweight` (`float[]`) – Concatenated lists of weights appearing in the weighted sums forming the $\overline{F}$ elements being specified. (input)

Groups

Problem data - affine expressions, *Problem data - semidefinite*

`Task.putafebarfrow`

```
putafebarfrow(afeidx,
              barvaridx,
              numterm,
              ptrterm,
              termidx,
              termweight)
```

This function inputs one row in $\overline{F}$. It first clears the row, i.e. sets $\overline{F}_{\text{afeidx},*} = 0$ and then sets the new entries. Each entry should be described as in *Task.putafebarfentry* and all those descriptions should be combined (for example concatenated) in the input to this method. That means the k-th entry set will have row index `afeidx`, symmetric variable index `barvaridx[k]` and the description of this term consists of indices in E and weights appearing in positions

$$\text{ptrterm}[k], \ldots, \text{ptrterm}[k] + (\text{numterm}[k] - 1)$$

in the corresponding arrays `termidx` and `termweight`. See *Task.putafebarfentry* for details.

Parameters

- `afeidx` (`int64`) – Row index of $\overline{F}$. (input)
- `barvaridx` (`int32[]`) – Semidefinite variable indexes. (input)
- `numterm` (`int64[]`) – The number of terms in the weighted sums that form each entry. (input)
- `ptrterm` (`int64[]`) – The pointer to the beginning of the description of each entry. (input)

- `termidx` (`int64[]`) – Concatenated lists of indices in E of the matrices appearing in the weighted sums for the $\overline{F}$ entries in the row. (input)
- `termweight` (`float[]`) – Concatenated lists of weights appearing in the weighted sums forming the $\overline{F}$ entries in the row. (input)

Groups

Problem data - affine expressions, *Problem data - semidefinite*

`Task.putafefcol`

```
putafefcol(varidx,afeidx,val)
```

Change one column of the matrix F of affine expressions. Resets all the elements in column `varidx` to zero and then sets

$$F_{\texttt{afeidx}[k],\texttt{varidx}} = \texttt{val}[k], \quad k = 0, \ldots, \texttt{numnz} - 1.$$

Parameters

- `varidx` (`int32`) – Index of a column in F. (input)
- `afeidx` (`int64[]`) – Row indexes of non-zero values in the column of F. (input)
- `val` (`float[]`) – New non-zero values in the column of F. (input)

Groups

Problem data - affine expressions

`Task.putafefentry`

```
putafefentry(afeidx,varidx,value)
```

Replaces one entry in the affine expression store F, that is it sets:

$$F_{\text{afeidx,varidx}} = \text{value}.$$

Parameters

- `afeidx` (`int64`) – Row index in F. (input)
- `varidx` (`int32`) – Column index in F. (input)
- `value` (`float`) – Value of $F_{\text{afeidx,varidx}}$. (input)

Groups

Problem data - affine expressions

`Task.putafefentrylist`

```
putafefentrylist(afeidx,varidx,val)
```

Replaces a number of entries in the affine expression store F, that is it sets:

$$F_{\text{afeidxs}[k],\text{varidx}[k]} = \text{val}[k]$$

for all k.

Parameters

- `afeidx` (`int64[]`) – Row indices in F. (input)
- `varidx` (`int32[]`) – Column indices in F. (input)
- `val` (`float[]`) – Values of the entries in F. (input)

Groups

Problem data - affine expressions

`Task.putafefrow`

```
putafefrow(afeidx,varidx,val)
```

Change one row of the matrix F of affine expressions. Resets all the elements in row `afeidx` to zero and then sets

$$F_{\texttt{afeidx},\texttt{varidx}[k]} = \texttt{val}[k], \quad k = 0, \ldots, \texttt{numnz} - 1.$$

Parameters

- `afeidx` (`int64`) – Index of a row in F. (input)
- `varidx` (`int32[]`) – Column indexes of non-zero values in the row of F. (input)
- `val` (`float[]`) – New non-zero values in the row of F. (input)

Groups

Problem data - affine expressions

`Task.putafefrowlist`

```
putafefrowlist(afeidx,numnzrow,ptrrow,varidx,val)
```

Clears and then changes a number of rows of the matrix F of affine expressions. The k-th of the rows to be changed has index $i = \text{afeidx}[k]$, contains $\text{numnzrow}[k]$ nonzeros and its description as in *Task.putafefrow* starts in position $\text{ptrrow}[k]$ of the arrays `varidx` and `val`. Formally, the row with index i is cleared and then set as:

$$F_{i,\text{varidx}[\text{ptrrow}[k]+j]} = \text{val}[\text{ptrrow}[k] + j], \quad j = 0, \ldots, \text{numnzrow}[k] - 1.$$

Parameters

- `afeidx` (`int64[]`) – Indices of rows in F. (input)
- `numnzrow` (`int32[]`) – Number of non-zeros in each of the modified rows of F. (input)
- `ptrrow` (`int64[]`) – Pointer to the first nonzero in each row of F. (input)
- `varidx` (`int32[]`) – Column indexes of non-zero values. (input)
- `val` (`float[]`) – New non-zero values in the rows of F. (input)

Groups

Problem data - affine expressions

`Task.putafeg`

```
putafeg(afeidx,g)
```

Change one element of the vector g in affine expressions i.e.

$$g_{\texttt{afeidx}} = \texttt{gi}.$$

Parameters

- `afeidx` (`int64`) – Index of an entry in g. (input)
- `g` (`float`) – New value for g_{afeidx}. (input)

Groups

Problem data - affine expressions

`Task.putafeglist`

```
putafeglist(afeidx,g)
```

Changes a list of elements of the vector g in affine expressions i.e. for all k it sets

$$g_{\text{afeidx}[k]} = \text{glist}[k].$$

Parameters

- `afeidx` (`int64[]`) – Indices of entries in g. (input)
- `g` (`float[]`) – New values for g. (input)

Groups

Problem data - affine expressions

`Task.putafegslice`

```
putafegslice(first,last,slice)
```

Modifies a slice in the vector g of constant terms in affine expressions using the principle

$$g_{\mathtt{j}} = \mathtt{slice}[\mathtt{j} - \mathtt{first}], \quad j = \mathrm{first}, .., \mathrm{last} - 1$$

Parameters

- `first` (`int64`) – First index in the sequence. (input)
- `last` (`int64`) – Last index plus 1 in the sequence. (input)
- `slice` (`float[]`) – The slice of g as a dense vector. The length is `last-first`. (input)

Groups

Problem data - affine expressions

`Task.putaij`

```
putaij(i,j,aij)
```

Changes a coefficient in the linear coefficient matrix A using the method

$$a_{i,j} = \mathtt{aij}.$$

Parameters

- `i` (`int32`) – Constraint (row) index. (input)
- `j` (`int32`) – Variable (column) index. (input)
- `aij` (`float`) – New coefficient for $a_{i,j}$. (input)

Groups

Problem data - linear part

`Task.putaijlist`

```
putaijlist(subi,subj,valij)
```

Changes one or more coefficients in A using the method

$$a_{\mathtt{subi}[\mathtt{k}],\mathtt{subj}[\mathtt{k}]} = \mathtt{valij}[\mathtt{k}], \quad k = 0, \ldots, \mathtt{num} - 1.$$

Duplicates are not allowed.

Parameters

- `subi` (`int32[]`) – Constraint (row) indices. (input)
- `subj` (`int32[]`) – Variable (column) indices. (input)
- `valij` (`float[]`) – New coefficient values for $a_{i,j}$. (input)

Groups

Problem data - linear part

`Task.putarow`

```
putarow(i,subi,vali)
```

Change one row of the linear constraint matrix A. Resets all the elements in row i to zero and then sets

$$a_{\texttt{i},\texttt{subi}[k]} = \texttt{vali}[k], \quad k = 0, \ldots, \texttt{nzi} - 1.$$

Parameters

- `i` (`int32`) – Index of a row in A. (input)
- `subi` (`int32[]`) – Column indexes of non-zero values in row i of A. (input)
- `vali` (`float[]`) – New non-zero values of row i in A. (input)

Groups

Problem data - linear part

`Task.putarowlist`

```
putarowlist(sub,ptrb,ptre,asub,aval)
```

Change a set of rows in the linear constraint matrix A with data in sparse triplet format. The requested rows are set to zero and then updated with:

$$\begin{array}{ll} \texttt{for} & i = 0, \ldots, \texttt{num} - 1 \\ & a_{\texttt{sub}[i],\texttt{asub}[k]} = \texttt{aval}[k], \quad k = \texttt{ptrb}[i], \ldots, \texttt{ptre}[i] - 1. \end{array}$$

Parameters

- `sub` (`int32[]`) – Indexes of rows that should be replaced, no duplicates. (input)
- `ptrb` (`int64[]`) – Array of pointers to the first element in each row. (input)
- `ptre` (`int64[]`) – Array of pointers to the last element plus one in each row. (input)
- `asub` (`int32[]`) – Column indexes of new elements. (input)
- `aval` (`float[]`) – Coefficient values. (input)

Groups

Problem data - linear part

`Task.putarowslice`

```
putarowslice(first,last,ptrb,ptre,asub,aval)
```

Change a slice of rows in the linear constraint matrix A with data in sparse triplet format. The requested rows are set to zero and then updated with:

$$\begin{array}{ll} \texttt{for} & i = \texttt{first}, \ldots, \texttt{last} - 1 \\ & a_{i,\texttt{asub}[k]} = \texttt{aval}[k], \quad k = \texttt{ptrb}[i - \texttt{first}], \ldots, \texttt{ptre}[i - \texttt{first}] - 1. \end{array}$$

Parameters

- `first` (`int32`) – First row in the slice. (input)
- `last` (`int32`) – Last row plus one in the slice. (input)
- `ptrb` (`int64[]`) – Array of pointers to the first element in each row. (input)
- `ptre` (`int64[]`) – Array of pointers to the last element plus one in each row. (input)
- `asub` (`int32[]`) – Column indexes of new elements. (input)
- `aval` (`float[]`) – Coefficient values. (input)

Groups

Problem data - linear part

`Task.putatruncatetol`

```
putatruncatetol(tolzero)
```

Truncates (sets to zero) all elements in A that satisfy

$$|a_{i,j}| \leq \texttt{tolzero}.$$

Parameters

`tolzero` (`float`) – Truncation tolerance. (input)

Groups

Problem data - linear part

`Task.putbarablocktriplet`

```
putbarablocktriplet(subi,subj,subk,subl,valijkl)
```

Inputs the $\overline{A}$ matrix in block triplet form.

Parameters

- `subi` (`int32[]`) – Constraint index. (input)
- `subj` (`int32[]`) – Symmetric matrix variable index. (input)
- `subk` (`int32[]`) – Block row index. (input)
- `subl` (`int32[]`) – Block column index. (input)
- `valijkl` (`float[]`) – The numerical value associated with each block triplet. (input)

Groups

Problem data - semidefinite

`Task.putbaraij`

```
putbaraij(i,j,sub,weights)
```

This function sets one element in the $\overline{A}$ matrix.

Each element in the $\overline{A}$ matrix is a weighted sum of symmetric matrices from the symmetric matrix storage E, so $\overline{A}_{ij}$ is a symmetric matrix. By default all elements in $\overline{A}$ are 0, so only non-zero elements need be added. Setting the same element again will overwrite the earlier entry.

The symmetric matrices from E are defined separately using the function *Task.appendsparsesymmat*.

Parameters

- `i` (`int32`) – Row index of $\overline{A}$. (input)
- `j` (`int32`) – Column index of $\overline{A}$. (input)
- `sub` (`int64[]`) – Indices in E of the matrices appearing in the weighted sum for $\overline{A}_{ij}$. (input)
- `weights` (`float[]`) – `weights[k]` is the coefficient of the `sub[k]`-th element of E in the weighted sum forming $\overline{A}_{ij}$. (input)

Groups

Problem data - semidefinite

`Task.putbaraijlist`

```
putbaraijlist(subi,
              subj,
              alphaptrb,
              alphaptre,
              matidx,
              weights)
```

This function sets a list of elements in the $\overline{A}$ matrix.

Each element in the $\overline{A}$ matrix is a weighted sum of symmetric matrices from the symmetric matrix storage E, so $\overline{A}_{ij}$ is a symmetric matrix. By default all elements in $\overline{A}$ are 0, so only non-zero elements need be added. Setting the same element again will overwrite the earlier entry.

The symmetric matrices from E are defined separately using the function *Task.appendsparsesymmat*.

Parameters

- `subi` (`int32[]`) – Row index of $\overline{A}$. (input)
- `subj` (`int32[]`) – Column index of $\overline{A}$. (input)
- `alphaptrb` (`int64[]`) – Start entries for terms in the weighted sum that forms $\overline{A}_{ij}$. (input)
- `alphaptre` (`int64[]`) – End entries for terms in the weighted sum that forms $\overline{A}_{ij}$. (input)
- `matidx` (`int64[]`) – Indices in E of the matrices appearing in the weighted sum for $\overline{A}_{ij}$. (input)
- `weights` (`float[]`) – `weights[k]` is the coefficient of the `sub[k]`-th element of E in the weighted sum forming $\overline{A}_{ij}$. (input)

Groups

Problem data - semidefinite

`Task.putbararowlist`

```
putbararowlist(subi,
               ptrb,
               ptre,
               subj,
               nummat,
               matidx,
               weights)
```

This function replaces a list of rows in the $\overline{A}$ matrix.

Parameters

- `subi` (`int32[]`) – Row indexes of $\overline{A}$. (input)
- `ptrb` (`int64[]`) – Start of rows in $\overline{A}$. (input)
- `ptre` (`int64[]`) – End of rows in $\overline{A}$. (input)
- `subj` (`int32[]`) – Column index of $\overline{A}$. (input)
- `nummat` (`int64[]`) – Number of entries in weighted sum of matrixes. (input)
- `matidx` (`int64[]`) – Matrix indexes for weighted sum of matrixes. (input)
- `weights` (`float[]`) – Weights for weighted sum of matrixes. (input)

Groups

Problem data - semidefinite

`Task.putbarcblocktriplet`

```
putbarcblocktriplet(subj,subk,subl,valjkl)
```

Inputs the $\overline{C}$ matrix in block triplet form.

Parameters

- `subj` (`int32[]`) – Symmetric matrix variable index. (input)
- `subk` (`int32[]`) – Block row index. (input)
- `subl` (`int32[]`) – Block column index. (input)
- `valjkl` (`float[]`) – The numerical value associated with each block triplet. (input)

Groups

Problem data - semidefinite

`Task.putbarcj`

```
putbarcj(j,sub,weights)
```

This function sets one entry in the $\overline{C}$ vector.

Each element in the $\overline{C}$ vector is a weighted sum of symmetric matrices from the symmetric matrix storage E, so $\overline{C}_j$ is a symmetric matrix. By default all elements in $\overline{C}$ are 0, so only non-zero elements need be added. Setting the same element again will overwrite the earlier entry.

The symmetric matrices from E are defined separately using the function *Task.appendsparsesymmat*.

Parameters

- `j` (`int32`) – Index of the element in $\overline{C}$ that should be changed. (input)
- `sub` (`int64[]`) – Indices in E of matrices appearing in the weighted sum for $\overline{C}_j$ (input)
- `weights` (`float[]`) – `weights[k]` is the coefficient of the `sub[k]`-th element of E in the weighted sum forming $\overline{C}_j$. (input)

Groups

Problem data - semidefinite, *Problem data - objective*

`Task.putbarsj`

```
putbarsj(whichsol,j,barsj)
```

Sets the dual solution for a semidefinite variable.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `j` (`int32`) – Index of the semidefinite variable. (input)
- `barsj` (`float[]`) – Value of $\overline{S}_j$. Format as in *Task.getbarsj*. (input)

Groups

Solution - semidefinite

`Task.putbarvarname`

```
putbarvarname(j,name)
```

Sets the name of a semidefinite variable.

Parameters

- `j` (`int32`) – Index of the variable. (input)
- `name` (`str`) – The variable name. (input)

Groups

Names, *Problem data - semidefinite*

`Task.putbarxj`

```
putbarxj(whichsol,j,barxj)
```

Sets the primal solution for a semidefinite variable.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `j` (`int32`) – Index of the semidefinite variable. (input)
- `barxj` (`float[]`) – Value of $\overline{X}_j$. Format as in *Task.getbarxj*. (input)

Groups

Solution - semidefinite

`Task.putcfix`

```
putcfix(cfix)
```

Replaces the fixed term in the objective by a new one.

Parameters

`cfix` (`float`) – Fixed term in the objective. (input)

Groups

Problem data - linear part, *Problem data - objective*

`Task.putcj`

```
putcj(j,cj)
```

Modifies one coefficient in the linear objective vector c, i.e.

$$c_{\texttt{j}} = \texttt{cj}.$$

If the absolute value exceeds *dparam.data_tol_c_huge* an error is generated. If the absolute value exceeds *dparam.data_tol_cj_large*, a warning is generated, but the coefficient is inputted as specified.

Parameters

- `j` (`int32`) – Index of the variable for which c should be changed. (input)
- `cj` (`float`) – New value of c_j. (input)

Groups

Problem data - linear part, *Problem data - objective*

`Task.putclist`

```
putclist(subj,val)
```

Modifies the coefficients in the linear term c in the objective using the principle

$$c_{\texttt{subj[t]}} = \texttt{val[t]}, \quad t = 0, \ldots, \texttt{num} - 1.$$

If a variable index is specified multiple times in `subj` only the last entry is used. Data checks are performed as in *Task.putcj*.

Parameters

- `subj` (`int32[]`) – Indices of variables for which the coefficient in c should be changed. (input)
- `val` (`float[]`) – New numerical values for coefficients in c that should be modified. (input)

Groups

Problem data - linear part, *Problem data - variables*, *Problem data - objective*

`Task.putconbound`

```
putconbound(i,bkc,blc,buc)
```

Changes the bounds for one constraint.

If the bound value specified is numerically larger than *dparam.data_tol_bound_inf* it is considered infinite and the bound key is changed accordingly. If a bound value is numerically larger than *dparam.data_tol_bound_wrn*, a warning will be displayed, but the bound is inputted as specified.

Parameters

- `i` (`int32`) – Index of the constraint. (input)
- `bkc` (*mosek.boundkey*) – New bound key. (input)

- `blc` (`float`) – New lower bound. (input)
- `buc` (`float`) – New upper bound. (input)

Groups

Problem data - linear part, *Problem data - constraints*, *Problem data - bounds*

`Task.putconboundlist`

```
putconboundlist(sub,bkc,blc,buc)
```

Changes the bounds for a list of constraints. If multiple bound changes are specified for a constraint, then only the last change takes effect. Data checks are performed as in *Task.putconbound*.

Parameters

- `sub` (`int32[]`) – List of constraint indexes. (input)
- `bkc` (*mosek.boundkey* `[]`) – Bound keys for the constraints. (input)
- `blc` (`float[]`) – Lower bounds for the constraints. (input)
- `buc` (`float[]`) – Upper bounds for the constraints. (input)

Groups

Problem data - linear part, *Problem data - constraints*, *Problem data - bounds*

`Task.putconboundlistconst`

```
putconboundlistconst(sub,bkc,blc,buc)
```

Changes the bounds for one or more constraints. Data checks are performed as in *Task.putconbound*.

Parameters

- `sub` (`int32[]`) – List of constraint indexes. (input)
- `bkc` (*mosek.boundkey*) – New bound key for all constraints in the list. (input)
- `blc` (`float`) – New lower bound for all constraints in the list. (input)
- `buc` (`float`) – New upper bound for all constraints in the list. (input)

Groups

Problem data - linear part, *Problem data - constraints*, *Problem data - bounds*

`Task.putconboundslice`

```
putconboundslice(first,last,bkc,blc,buc)
```

Changes the bounds for a slice of the constraints. Data checks are performed as in *Task.putconbound*.

Parameters

- `first` (`int32`) – First index in the sequence. (input)
- `last` (`int32`) – Last index plus 1 in the sequence. (input)
- `bkc` (*mosek.boundkey* `[]`) – Bound keys for the constraints. (input)
- `blc` (`float[]`) – Lower bounds for the constraints. (input)
- `buc` (`float[]`) – Upper bounds for the constraints. (input)

Groups

Problem data - linear part, *Problem data - constraints*, *Problem data - bounds*

`Task.putconboundsliceconst`

```
putconboundsliceconst(first,last,bkc,blc,buc)
```

Changes the bounds for a slice of the constraints. Data checks are performed as in *Task.putconbound*.

Parameters

- `first` (`int32`) – First index in the sequence. (input)
- `last` (`int32`) – Last index plus 1 in the sequence. (input)
- `bkc` (*mosek.boundkey*) – New bound key for all constraints in the slice. (input)
- `blc` (`float`) – New lower bound for all constraints in the slice. (input)
- `buc` (`float`) – New upper bound for all constraints in the slice. (input)

Groups

Problem data - linear part, *Problem data - constraints*, *Problem data - bounds*

~~`Task.putcone`~~ *Deprecated*

```
putcone(k,ct,conepar,submem)
```

NOTE: This interface to conic optimization is deprecated and will be removed in a future major release. Conic problems should be specified using the affine conic constraints interface (ACC), see Sec. 6.2 for details.

Parameters

- `k` (`int32`) – Index of the cone. (input)
- `ct` (*mosek.conetype*) – Specifies the type of the cone. (input)
- `conepar` (`float`) – For the power cone it denotes the exponent alpha. For other cone types it is unused and can be set to 0. (input)
- `submem` (`int32[]`) – Variable subscripts of the members in the cone. (input)

Groups

Problem data - cones (deprecated)

~~`Task.putconename`~~ *Deprecated*

```
putconename(j,name)
```

NOTE: This interface to conic optimization is deprecated and will be removed in a future major release. Conic problems should be specified using the affine conic constraints interface (ACC), see Sec. 6.2 for details.

Parameters

- `j` (`int32`) – Index of the cone. (input)
- `name` (`str`) – The name of the cone. (input)

Groups

Names, *Problem data - cones (deprecated)*

`Task.putconname`

```
putconname(i,name)
```

Sets the name of a constraint.

Parameters

- `i` (`int32`) – Index of the constraint. (input)
- `name` (`str`) – The name of the constraint. (input)

Groups

Names, *Problem data - constraints*, *Problem data - linear part*

`Task.putconsolutioni`

```
putconsolutioni(i,whichsol,sk,x,sl,su)
```

Sets the primal and dual solution information for a single constraint.

Parameters

- `i` (`int32`) – Index of the constraint. (input)

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `sk` (*mosek.stakey*) – Status key of the constraint. (input)
- `x` (`float`) – Primal solution value of the constraint. (input)
- `sl` (`float`) – Solution value of the dual variable associated with the lower bound. (input)
- `su` (`float`) – Solution value of the dual variable associated with the upper bound. (input)

Groups

Solution information, *Solution - primal*, *Solution - dual*

`Task.putcslice`

```
putcslice(first,last,slice)
```

Modifies a slice in the linear term c in the objective using the principle

$$c_{\mathtt{j}} = \mathtt{slice}[\mathtt{j} - \mathtt{first}], \quad j = \mathtt{first}, .., \mathtt{last} - 1$$

Data checks are performed as in *Task.putcj*.

Parameters

- `first` (`int32`) – First element in the slice of c. (input)
- `last` (`int32`) – Last element plus 1 of the slice in c to be changed. (input)
- `slice` (`float[]`) – New numerical values for coefficients in c that should be modified. (input)

Groups

Problem data - linear part, *Problem data - objective*

`Task.putdjc`

```
putdjc(djcidx,
       domidxlist,
       afeidxlist,
       b,
       termsizelist)
```

Inputs a disjunctive constraint. The constraint has the form

$$T_1 \text{ or } T_2 \text{ or } \cdots \text{ or } T_{\text{numterms}}$$

For each $i = 1, \ldots$ numterms the i-th clause (term) T_i has the form *a sequence of affine expressions belongs to a product of domains*, where the number of domains is termsizelist$[i]$ and the number of affine expressions is equal to the sum of dimensions of all domains appearing in T_i.

All the domains and all the affine expressions appearing in the above description are arranged sequentially in the lists `domidxlist` and `afeidxlist`, respectively. In particular, the length of `domidxlist` must be equal to the sum of elements of `termsizelist`, and the length of `afeidxlist` must be equal to the sum of dimensions of all the domains appearing in `domidxlist`.

The elements of `domidxlist` are indexes of domains previously defined with one of the `append...domain` functions.

The elements of `afeidxlist` are indexes to the store of affine expressions, i.e. the k-th affine expression appearing in the disjunctive constraint is going to be

$$F_{\text{afeidxlist}[k],:}x + g_{\text{afeidxlist}[k]}$$

If an optional vector `b` of the same length as `afeidxlist` is specified then the k-th affine expression appearing in the disjunctive constraint will be taken as

$$F_{\text{afeidxlist}[k],:}x + g_{\text{afeidxlist}[k]} - b_k$$

Parameters

- `djcidx` (`int64`) – Index of the disjunctive constraint. (input)
- `domidxlist` (`int64[]`) – List of domain indexes. (input)
- `afeidxlist` (`int64[]`) – List of affine expression indexes. (input)
- `b` (`float[]`) – The vector of constant terms modifying affine expressions. (input)
- `termsizelist` (`int64[]`) – List of term sizes. (input)

Groups

Problem data - disjunctive constraints

`Task.putdjcname`

```
putdjcname(djcidx,name)
```

Sets the name of a disjunctive constraint.

Parameters

- `djcidx` (`int64`) – Index of the disjunctive constraint. (input)
- `name` (`str`) – The name of the disjunctive constraint. (input)

Groups

Names, *Problem data - disjunctive constraints*

`Task.putdjcslice`

```
putdjcslice(idxfirst,
            idxlast,
            domidxlist,
            afeidxlist,
            b,
            termsizelist,
            termsindjc)
```

Inputs a slice of disjunctive constraints.

The array `termsindjc` should have length idxlast − idxfirst and contain the number of terms in consecutive constraints forming the slice.

The rest of the input consists of concatenated descriptions of individual constraints, where each constraint is described as in *`Task.putdjc`*.

Parameters

- `idxfirst` (`int64`) – Index of the first disjunctive constraint in the slice. (input)
- `idxlast` (`int64`) – Index of the last disjunctive constraint in the slice plus 1. (input)
- `domidxlist` (`int64[]`) – List of domain indexes. (input)
- `afeidxlist` (`int64[]`) – List of affine expression indexes. (input)
- `b` (`float[]`) – The vector of constant terms modifying affine expressions. Optional, can be `None` if not required. (input)
- `termsizelist` (`int64[]`) – List of term sizes. (input)
- `termsindjc` (`int64[]`) – Number of terms in each of the disjunctive constraints in the slice. (input)

Groups

Problem data - disjunctive constraints

`Task.putdomainname`

```
putdomainname(domidx,name)
```

Sets the name of a domain.

Parameters

- `domidx` (`int64`) – Index of the domain. (input)
- `name` (`str`) – The name of the domain. (input)

Groups
Names, *Problem data - domain*

`Task.putdouparam`

```
putdouparam(param,parvalue)
```

Sets the value of a double parameter.

Parameters
- `param` (*mosek.dparam*) – Which parameter. (input)
- `parvalue` (`float`) – Parameter value. (input)

Groups
Parameters

`Task.putintparam`

```
putintparam(param,parvalue)
```

Sets the value of an integer parameter.

Parameters
- `param` (*mosek.iparam*) – Which parameter. (input)
- `parvalue` (`int32`) – Parameter value. (input)

Groups
Parameters

`Task.putlintparam`

```
putlintparam(param,parvalue)
```

Sets the value of an integer parameter.

Parameters
- `param` (*mosek.iparam*) – Which parameter. (input)
- `parvalue` (`int64`) – Parameter value. (input)

Groups
Parameters

`Task.putmaxnumacc`

```
putmaxnumacc(maxnumacc)
```

Sets the number of preallocated affine conic constraints in the optimization task. When this number is reached **MOSEK** will automatically allocate more space. It is never mandatory to call this function, since **MOSEK** will reallocate any internal structures whenever it is required.

Parameters
`maxnumacc` (`int64`) – Number of preallocated affine conic constraints. (input)

Groups
Environment and task management, *Problem data - affine conic constraints*

`Task.putmaxnumafe`

```
putmaxnumafe(maxnumafe)
```

Sets the number of preallocated affine expressions in the optimization task. When this number is reached **MOSEK** will automatically allocate more space for affine expressions. It is never mandatory to call this function, since **MOSEK** will reallocate any internal structures whenever it is required.

Parameters
`maxnumafe` (`int64`) – Number of preallocated affine expressions. (input)

Groups
Environment and task management, *Problem data - affine expressions*

`Task.putmaxnumanz`

```
putmaxnumanz(maxnumanz)
```

Sets the number of preallocated non-zero entries in A.

MOSEK stores only the non-zero elements in the linear coefficient matrix A and it cannot predict how much storage is required to store A. Using this function it is possible to specify the number of non-zeros to preallocate for storing A.

If the number of non-zeros in the problem is known, it is a good idea to set `maxnumanz` slightly larger than this number, otherwise a rough estimate can be used. In general, if A is inputted in many small chunks, setting this value may speed up the data input phase.

It is not mandatory to call this function, since **MOSEK** will reallocate internal structures whenever it is necessary.

The function call has no effect if both `maxnumcon` and `maxnumvar` are zero.

Parameters
`maxnumanz` (`int64`) – Number of preallocated non-zeros in A. (input)

Groups
Environment and task management, *Problem data - linear part*

`Task.putmaxnumbarvar`

```
putmaxnumbarvar(maxnumbarvar)
```

Sets the number of preallocated symmetric matrix variables in the optimization task. When this number of variables is reached **MOSEK** will automatically allocate more space for variables.

It is not mandatory to call this function. It only gives a hint about the amount of data to preallocate for efficiency reasons.

Please note that `maxnumbarvar` must be larger than the current number of symmetric matrix variables in the task.

Parameters
`maxnumbarvar` (`int32`) – Number of preallocated symmetric matrix variables. (input)

Groups
Environment and task management, *Problem data - semidefinite*

`Task.putmaxnumcon`

```
putmaxnumcon(maxnumcon)
```

Sets the number of preallocated constraints in the optimization task. When this number of constraints is reached **MOSEK** will automatically allocate more space for constraints.

It is never mandatory to call this function, since **MOSEK** will reallocate any internal structures whenever it is required.

Please note that `maxnumcon` must be larger than the current number of constraints in the task.

Parameters

`maxnumcon` (`int32`) – Number of preallocated constraints in the optimization task. (input)

Groups

Environment and task management, *Problem data - constraints*

~~`Task.putmaxnumcone`~~ *Deprecated*

```
putmaxnumcone(maxnumcone)
```

NOTE: This interface to conic optimization is deprecated and will be removed in a future major release. Conic problems should be specified using the affine conic constraints interface (ACC), see Sec. 6.2 for details.

Sets the number of preallocated conic constraints in the optimization task. When this number of conic constraints is reached **MOSEK** will automatically allocate more space for conic constraints.

It is not mandatory to call this function, since **MOSEK** will reallocate any internal structures whenever it is required.

Please note that `maxnumcon` must be larger than the current number of conic constraints in the task.

Parameters

`maxnumcone` (`int32`) – Number of preallocated conic constraints in the optimization task. (input)

Groups

Environment and task management, *Problem data - cones (deprecated)*

`Task.putmaxnumdjc`

```
putmaxnumdjc(maxnumdjc)
```

Sets the number of preallocated disjunctive constraints in the optimization task. When this number is reached **MOSEK** will automatically allocate more space. It is never mandatory to call this function, since **MOSEK** will reallocate any internal structures whenever it is required.

Parameters

`maxnumdjc` (`int64`) – Number of preallocated disjunctive constraints in the task. (input)

Groups

Environment and task management, *Problem data - disjunctive constraints*

`Task.putmaxnumdomain`

```
putmaxnumdomain(maxnumdomain)
```

Sets the number of preallocated domains in the optimization task. When this number is reached **MOSEK** will automatically allocate more space. It is never mandatory to call this function, since **MOSEK** will reallocate any internal structures whenever it is required.

Parameters

`maxnumdomain` (`int64`) – Number of preallocated domains. (input)

Groups

Environment and task management, *Problem data - domain*

`Task.putmaxnumqnz`

```
putmaxnumqnz(maxnumqnz)
```

Sets the number of preallocated non-zero entries in quadratic terms.

MOSEK stores only the non-zero elements in Q. Therefore, **MOSEK** cannot predict how much storage is required to store Q. Using this function it is possible to specify the number non-zeros to preallocate for storing Q (both objective and constraints).

It may be advantageous to reserve more non-zeros for Q than actually needed since it may improve the internal efficiency of **MOSEK**, however, it is never worthwhile to specify more than the double of the anticipated number of non-zeros in Q.

It is not mandatory to call this function, since **MOSEK** will reallocate internal structures whenever it is necessary.

Parameters

`maxnumqnz` (`int64`) – Number of non-zero elements preallocated in quadratic coefficient matrices. (input)

Groups

Environment and task management, *Problem data - quadratic part*

`Task.putmaxnumvar`

```
putmaxnumvar(maxnumvar)
```

Sets the number of preallocated variables in the optimization task. When this number of variables is reached **MOSEK** will automatically allocate more space for variables.

It is not mandatory to call this function. It only gives a hint about the amount of data to preallocate for efficiency reasons.

Please note that `maxnumvar` must be larger than the current number of variables in the task.

Parameters

`maxnumvar` (`int32`) – Number of preallocated variables in the optimization task. (input)

Groups

Environment and task management, *Problem data - variables*

`Task.putnadouparam`

```
putnadouparam(paramname,parvalue)
```

Sets the value of a named double parameter.

Parameters

- `paramname` (`str`) – Name of a parameter. (input)
- `parvalue` (`float`) – Parameter value. (input)

Groups

Parameters

`Task.putnaintparam`

```
putnaintparam(paramname,parvalue)
```

Sets the value of a named integer parameter.

Parameters

- `paramname` (`str`) – Name of a parameter. (input)
- `parvalue` (`int32`) – Parameter value. (input)

Groups

Parameters

`Task.putnastrparam`

```
putnastrparam(paramname,parvalue)
```

Sets the value of a named string parameter.

Parameters

- `paramname` (`str`) – Name of a parameter. (input)
- `parvalue` (`str`) – Parameter value. (input)

Groups

Parameters

`Task.putobjname`

```
putobjname(objname)
```

Assigns a new name to the objective.

Parameters

`objname` (`str`) – Name of the objective. (input)

Groups

Problem data - linear part, *Names*, *Problem data - objective*

`Task.putobjsense`

```
putobjsense(sense)
```

Sets the objective sense of the task.

Parameters

`sense` (*mosek.objsense*) – The objective sense of the task. The values *objsense.maximize* and *objsense.minimize* mean that the problem is maximized or minimized respectively. (input)

Groups

Problem data - linear part, *Problem data - objective*

`Task.putoptserverhost`

```
putoptserverhost(host)
```

Specify an OptServer URL for remote calls. The URL should contain protocol, host and port in the form `http://server:port` or `https://server:port`. If the URL is set using this function, all subsequent calls to any **MOSEK** function that involves synchronous optimization will be sent to the specified OptServer instead of being executed locally. Passing `None` or empty string deactivates this redirection.

Has the same effect as setting the parameter *sparam.remote_optserver_host*.

Parameters

`host` (`str`) – A URL specifying the optimization server to be used. (input)

Groups

Remote optimization

`Task.putparam`

```
putparam(parname,parvalue)
```

Checks if `parname` is valid parameter name. If it is, the parameter is assigned the value specified by `parvalue`.

Parameters

- `parname` (`str`) – Parameter name. (input)
- `parvalue` (`str`) – Parameter value. (input)

Groups

Parameters

`Task.putqcon`

```
putqcon(qcsubk,qcsubi,qcsubj,qcval)
```

Replace all quadratic entries in the constraints. The list of constraints has the form

$$l_k^c \leq \frac{1}{2} \sum_{i=0}^{\texttt{numvar}-1} \sum_{j=0}^{\texttt{numvar}-1} q_{ij}^k x_i x_j + \sum_{j=0}^{\texttt{numvar}-1} a_{kj} x_j \leq u_k^c, \quad k = 0, \ldots, m-1.$$

This function sets all the quadratic terms to zero and then performs the update:

$$q^{\texttt{qcsubk[t]}}_{\texttt{qcsubi[t]},\texttt{qcsubj[t]}} = q^{\texttt{qcsubk[t]}}_{\texttt{qcsubj[t]},\texttt{qcsubi[t]}} = q^{\texttt{qcsubk[t]}}_{\texttt{qcsubj[t]},\texttt{qcsubi[t]}} + \texttt{qcval[t]},$$

for $t = 0, \ldots, \texttt{numqcnz} - 1$.

Please note that:

- For large problems it is essential for the efficiency that the function *Task.putmaxnumqnz* is employed to pre-allocate space.
- Only the lower triangular parts should be specified because the Q matrices are symmetric. Specifying entries where $i < j$ will result in an error.
- Only non-zero elements should be specified.
- The order in which the non-zero elements are specified is insignificant.
- Duplicate elements are added together as shown above. Hence, it is usually not recommended to specify the same entry multiple times.

For a code example see Section *Quadratic Optimization*

Parameters

- `qcsubk` (`int32[]`) – Constraint subscripts for quadratic coefficients. (input)
- `qcsubi` (`int32[]`) – Row subscripts for quadratic constraint matrix. (input)
- `qcsubj` (`int32[]`) – Column subscripts for quadratic constraint matrix. (input)
- `qcval` (`float[]`) – Quadratic constraint coefficient values. (input)

Groups

Problem data - quadratic part

`Task.putqconk`

```
putqconk(k,qcsubi,qcsubj,qcval)
```

Replaces all the quadratic entries in one constraint. This function performs the same operations as *Task.putqcon* but only with respect to constraint number `k` and it does not modify the other constraints. See the description of *Task.putqcon* for definitions and important remarks.

Parameters

- `k` (`int32`) – The constraint in which the new Q elements are inserted. (input)
- `qcsubi` (`int32[]`) – Row subscripts for quadratic constraint matrix. (input)
- `qcsubj` (`int32[]`) – Column subscripts for quadratic constraint matrix. (input)
- `qcval` (`float[]`) – Quadratic constraint coefficient values. (input)

Groups

Problem data - quadratic part

`Task.putqobj`

```
putqobj(qosubi,qosubj,qoval)
```

Replace all quadratic terms in the objective. If the objective has the form

$$\frac{1}{2}\sum_{i=0}^{\texttt{numvar}-1}\sum_{j=0}^{\texttt{numvar}-1} q^o_{ij}x_ix_j + \sum_{j=0}^{\texttt{numvar}-1} c_jx_j + c^f$$

then this function sets all the quadratic terms to zero and then performs the update:

$$q^o_{\texttt{qosubi[t]},\texttt{qosubj[t]}} = q^o_{\texttt{qosubj[t]},\texttt{qosubi[t]}} = q^o_{\texttt{qosubj[t]},\texttt{qosubi[t]}} + \texttt{qoval[t]},$$

for $t = 0, \ldots, \texttt{numqonz} - 1$.

See the description of *Task.putqcon* for important remarks and example.

Parameters

- `qosubi` (`int32[]`) – Row subscripts for quadratic objective coefficients. (input)
- `qosubj` (`int32[]`) – Column subscripts for quadratic objective coefficients. (input)
- `qoval` (`float[]`) – Quadratic objective coefficient values. (input)

Groups

Problem data - quadratic part, *Problem data - objective*

`Task.putqobjij`

```
putqobjij(i,j,qoij)
```

Replaces one coefficient in the quadratic term in the objective. The function performs the assignment

$$q^o_{ij} = q^o_{ji} = \texttt{qoij}.$$

Only the elements in the lower triangular part are accepted. Setting q_{ij} with $j > i$ will cause an error.

Please note that replacing all quadratic elements one by one is more computationally expensive than replacing them all at once. Use *Task.putqobj* instead whenever possible.

Parameters

- `i` (`int32`) – Row index for the coefficient to be replaced. (input)
- `j` (`int32`) – Column index for the coefficient to be replaced. (input)
- `qoij` (`float`) – The new value for q^o_{ij}. (input)

Groups

Problem data - quadratic part, *Problem data - objective*

`Task.putskc`

```
putskc(whichsol,skc)
```

Sets the status keys for the constraints.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `skc` (*mosek.stakey* `[]`) – Status keys for the constraints. (input)

Groups

Solution information

`Task.putskcslice`

```
putskcslice(whichsol,first,last,skc)
```

Sets the status keys for a slice of the constraints.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (`int32`) – First index in the sequence. (input)
- `last` (`int32`) – Last index plus 1 in the sequence. (input)
- `skc` (*mosek.stakey* `[]`) – Status keys for the constraints. (input)

Groups

Solution information

`Task.putskx`

```
putskx(whichsol,skx)
```

Sets the status keys for the scalar variables.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `skx` (*mosek.stakey* `[]`) – Status keys for the variables. (input)

Groups

Solution information

`Task.putskxslice`

```
putskxslice(whichsol,first,last,skx)
```

Sets the status keys for a slice of the variables.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (`int32`) – First index in the sequence. (input)
- `last` (`int32`) – Last index plus 1 in the sequence. (input)
- `skx` (*mosek.stakey* `[]`) – Status keys for the variables. (input)

Groups

Solution information

`Task.putslc`

```
putslc(whichsol,slc)
```

Sets the s_l^c vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `slc` (`float[]`) – Dual variables corresponding to the lower bounds on the constraints. (input)

Groups

Solution - dual

`Task.putslcslice`

```
putslcslice(whichsol,first,last,slc)
```

Sets a slice of the s_l^c vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)

- `first` (`int32`) – First index in the sequence. (input)
- `last` (`int32`) – Last index plus 1 in the sequence. (input)
- `slc` (`float[]`) – Dual variables corresponding to the lower bounds on the constraints. (input)

Groups

Solution - dual

`Task.putslx`

```
putslx(whichsol,slx)
```

Sets the s_l^x vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `slx` (`float[]`) – Dual variables corresponding to the lower bounds on the variables. (input)

Groups

Solution - dual

`Task.putslxslice`

```
putslxslice(whichsol,first,last,slx)
```

Sets a slice of the s_l^x vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (`int32`) – First index in the sequence. (input)
- `last` (`int32`) – Last index plus 1 in the sequence. (input)
- `slx` (`float[]`) – Dual variables corresponding to the lower bounds on the variables. (input)

Groups

Solution - dual

`Task.putsnx`

```
putsnx(whichsol,sux)
```

Sets the s_n^x vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `sux` (`float[]`) – Dual variables corresponding to the upper bounds on the variables. (input)

Groups

Solution - dual

`Task.putsnxslice`

```
putsnxslice(whichsol,first,last,snx)
```

Sets a slice of the s_n^x vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (`int32`) – First index in the sequence. (input)
- `last` (`int32`) – Last index plus 1 in the sequence. (input)

- `snx` (`float[]`) – Dual variables corresponding to the conic constraints on the variables. (input)

Groups

Solution - dual

`Task.putsolution`

```
putsolution(whichsol,
            skc,
            skx,
            skn,
            xc,
            xx,
            y,
            slc,
            suc,
            slx,
            sux,
            snx)
```

Inserts a solution into the task.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `skc` (*mosek.stakey* `[]`) – Status keys for the constraints. (input)
- `skx` (*mosek.stakey* `[]`) – Status keys for the variables. (input)
- `skn` (*mosek.stakey* `[]`) – Status keys for the conic constraints. (input)
- `xc` (`float[]`) – Primal constraint solution. (input)
- `xx` (`float[]`) – Primal variable solution. (input)
- `y` (`float[]`) – Vector of dual variables corresponding to the constraints. (input)
- `slc` (`float[]`) – Dual variables corresponding to the lower bounds on the constraints. (input)
- `suc` (`float[]`) – Dual variables corresponding to the upper bounds on the constraints. (input)
- `slx` (`float[]`) – Dual variables corresponding to the lower bounds on the variables. (input)
- `sux` (`float[]`) – Dual variables corresponding to the upper bounds on the variables. (input)
- `snx` (`float[]`) – Dual variables corresponding to the conic constraints on the variables. (input)

Groups

Solution information, *Solution - primal*, *Solution - dual*

`Task.putsolutionnew`

```
putsolutionnew(whichsol,
               skc,
               skx,
               skn,
               xc,
               xx,
               y,
               slc,
               suc,
               slx,
               sux,
               snx,
               doty)
```

Inserts a solution into the task.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `skc` (*mosek.stakey* `[]`) – Status keys for the constraints. (input)
- `skx` (*mosek.stakey* `[]`) – Status keys for the variables. (input)
- `skn` (*mosek.stakey* `[]`) – Status keys for the conic constraints. (input)
- `xc` (`float[]`) – Primal constraint solution. (input)
- `xx` (`float[]`) – Primal variable solution. (input)
- `y` (`float[]`) – Vector of dual variables corresponding to the constraints. (input)
- `slc` (`float[]`) – Dual variables corresponding to the lower bounds on the constraints. (input)
- `suc` (`float[]`) – Dual variables corresponding to the upper bounds on the constraints. (input)
- `slx` (`float[]`) – Dual variables corresponding to the lower bounds on the variables. (input)
- `sux` (`float[]`) – Dual variables corresponding to the upper bounds on the variables. (input)
- `snx` (`float[]`) – Dual variables corresponding to the conic constraints on the variables. (input)
- `doty` (`float[]`) – Dual variables corresponding to affine conic constraints. (input)

Groups

Solution information, *Solution - primal*, *Solution - dual*

`Task.putsolutionyi`

```
putsolutionyi(i,whichsol,y)
```

Inputs the dual variable of a solution.

Parameters

- `i` (`int32`) – Index of the dual variable. (input)
- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `y` (`float`) – Solution value of the dual variable. (input)

Groups

Solution information, *Solution - dual*

`Task.putstrparam`

```
putstrparam(param,parvalue)
```

Sets the value of a string parameter.

Parameters

- `param` (*mosek.sparam*) – Which parameter. (input)
- `parvalue` (`str`) – Parameter value. (input)

Groups

Parameters

`Task.putsuc`

```
putsuc(whichsol,suc)
```

Sets the s_u^c vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)

- `suc` (`float[]`) – Dual variables corresponding to the upper bounds on the constraints. (input)

Groups
Solution - dual

`Task.putsucslice`

```
putsucslice(whichsol,first,last,suc)
```

Sets a slice of the s_u^c vector for a solution.

Parameters
- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (`int32`) – First index in the sequence. (input)
- `last` (`int32`) – Last index plus 1 in the sequence. (input)
- `suc` (`float[]`) – Dual variables corresponding to the upper bounds on the constraints. (input)

Groups
Solution - dual

`Task.putsux`

```
putsux(whichsol,sux)
```

Sets the s_u^x vector for a solution.

Parameters
- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `sux` (`float[]`) – Dual variables corresponding to the upper bounds on the variables. (input)

Groups
Solution - dual

`Task.putsuxslice`

```
putsuxslice(whichsol,first,last,sux)
```

Sets a slice of the s_u^x vector for a solution.

Parameters
- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (`int32`) – First index in the sequence. (input)
- `last` (`int32`) – Last index plus 1 in the sequence. (input)
- `sux` (`float[]`) – Dual variables corresponding to the upper bounds on the variables. (input)

Groups
Solution - dual

`Task.puttaskname`

```
puttaskname(taskname)
```

Assigns a new name to the task.

Parameters
`taskname` (`str`) – Name assigned to the task. (input)

Groups
Names, *Environment and task management*

`Task.putvarbound`

```
putvarbound(j,bkx,blx,bux)
```

Changes the bounds for one variable.

If the bound value specified is numerically larger than *dparam.data_tol_bound_inf* it is considered infinite and the bound key is changed accordingly. If a bound value is numerically larger than *dparam.data_tol_bound_wrn*, a warning will be displayed, but the bound is inputted as specified.

Parameters

- `j` (`int32`) – Index of the variable. (input)
- `bkx` (*mosek.boundkey*) – New bound key. (input)
- `blx` (`float`) – New lower bound. (input)
- `bux` (`float`) – New upper bound. (input)

Groups

Problem data - linear part, *Problem data - variables*, *Problem data - bounds*

`Task.putvarboundlist`

```
putvarboundlist(sub,bkx,blx,bux)
```

Changes the bounds for one or more variables. If multiple bound changes are specified for a variable, then only the last change takes effect. Data checks are performed as in *Task.putvarbound*.

Parameters

- `sub` (`int32[]`) – List of variable indexes. (input)
- `bkx` (*mosek.boundkey* `[]`) – Bound keys for the variables. (input)
- `blx` (`float[]`) – Lower bounds for the variables. (input)
- `bux` (`float[]`) – Upper bounds for the variables. (input)

Groups

Problem data - linear part, *Problem data - variables*, *Problem data - bounds*

`Task.putvarboundlistconst`

```
putvarboundlistconst(sub,bkx,blx,bux)
```

Changes the bounds for one or more variables. Data checks are performed as in *Task.putvarbound*.

Parameters

- `sub` (`int32[]`) – List of variable indexes. (input)
- `bkx` (*mosek.boundkey*) – New bound key for all variables in the list. (input)
- `blx` (`float`) – New lower bound for all variables in the list. (input)
- `bux` (`float`) – New upper bound for all variables in the list. (input)

Groups

Problem data - linear part, *Problem data - variables*, *Problem data - bounds*

`Task.putvarboundslice`

```
putvarboundslice(first,last,bkx,blx,bux)
```

Changes the bounds for a slice of the variables. Data checks are performed as in *Task.putvarbound*.

Parameters

- `first` (`int32`) – First index in the sequence. (input)
- `last` (`int32`) – Last index plus 1 in the sequence. (input)
- `bkx` (*mosek.boundkey* `[]`) – Bound keys for the variables. (input)
- `blx` (`float[]`) – Lower bounds for the variables. (input)

- `bux` (`float[]`) – Upper bounds for the variables. (input)

Groups

Problem data - linear part, *Problem data - variables*, *Problem data - bounds*

`Task.putvarboundsliceconst`

```
putvarboundsliceconst(first,last,bkx,blx,bux)
```

Changes the bounds for a slice of the variables. Data checks are performed as in *Task.putvarbound*.

Parameters

- `first` (`int32`) – First index in the sequence. (input)
- `last` (`int32`) – Last index plus 1 in the sequence. (input)
- `bkx` (*mosek.boundkey*) – New bound key for all variables in the slice. (input)
- `blx` (`float`) – New lower bound for all variables in the slice. (input)
- `bux` (`float`) – New upper bound for all variables in the slice. (input)

Groups

Problem data - linear part, *Problem data - variables*, *Problem data - bounds*

`Task.putvarname`

```
putvarname(j,name)
```

Sets the name of a variable.

Parameters

- `j` (`int32`) – Index of the variable. (input)
- `name` (`str`) – The variable name. (input)

Groups

Names, *Problem data - variables*, *Problem data - linear part*

`Task.putvarsolutionj`

```
putvarsolutionj(j,whichsol,sk,x,sl,su,sn)
```

Sets the primal and dual solution information for a single variable.

Parameters

- `j` (`int32`) – Index of the variable. (input)
- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `sk` (*mosek.stakey*) – Status key of the variable. (input)
- `x` (`float`) – Primal solution value of the variable. (input)
- `sl` (`float`) – Solution value of the dual variable associated with the lower bound. (input)
- `su` (`float`) – Solution value of the dual variable associated with the upper bound. (input)
- `sn` (`float`) – Solution value of the dual variable associated with the conic constraint. (input)

Groups

Solution information, *Solution - primal*, *Solution - dual*

`Task.putvartype`

```
putvartype(j,vartype)
```

Sets the variable type of one variable.

Parameters

- `j` (`int32`) – Index of the variable. (input)
- `vartype` (*`mosek.variabletype`*) – The new variable type. (input)

Groups

Problem data - variables

`Task.putvartypelist`

```
putvartypelist(subj,vartype)
```

Sets the variable type for one or more variables. If the same index is specified multiple times in `subj` only the last entry takes effect.

Parameters

- `subj` (`int32[]`) – A list of variable indexes for which the variable type should be changed. (input)
- `vartype` (*`mosek.variabletype`*`[]`) – A list of variable types that should be assigned to the variables specified by `subj`. (input)

Groups

Problem data - variables

`Task.putxc`

```
putxc(whichsol,xc)
```

```
putxc(whichsol) -> (xc)
```

Sets the x^c vector for a solution.

Parameters

- `whichsol` (*`mosek.soltype`*) – Selects a solution. (input)
- `xc` (`float[]`) – Primal constraint solution. (output)

Return

`xc` (`float[]`) – Primal constraint solution.

Groups

Solution - primal

`Task.putxcslice`

```
putxcslice(whichsol,first,last,xc)
```

Sets a slice of the x^c vector for a solution.

Parameters

- `whichsol` (*`mosek.soltype`*) – Selects a solution. (input)
- `first` (`int32`) – First index in the sequence. (input)
- `last` (`int32`) – Last index plus 1 in the sequence. (input)
- `xc` (`float[]`) – Primal constraint solution. (input)

Groups

Solution - primal

`Task.putxx`

```
putxx(whichsol,xx)
```

Sets the x^x vector for a solution.

Parameters

- `whichsol` (*`mosek.soltype`*) – Selects a solution. (input)
- `xx` (`float[]`) – Primal variable solution. (input)

Groups
Solution - primal

`Task.putxxslice`

```
putxxslice(whichsol,first,last,xx)
```

Sets a slice of the x^x vector for a solution.

Parameters
- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (`int32`) – First index in the sequence. (input)
- `last` (`int32`) – Last index plus 1 in the sequence. (input)
- `xx` (`float[]`) – Primal variable solution. (input)

Groups
Solution - primal

`Task.puty`

```
puty(whichsol,y)
```

Sets the y vector for a solution.

Parameters
- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `y` (`float[]`) – Vector of dual variables corresponding to the constraints. (input)

Groups
Solution - primal

`Task.putyslice`

```
putyslice(whichsol,first,last,y)
```

Sets a slice of the y vector for a solution.

Parameters
- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (`int32`) – First index in the sequence. (input)
- `last` (`int32`) – Last index plus 1 in the sequence. (input)
- `y` (`float[]`) – Vector of dual variables corresponding to the constraints. (input)

Groups
Solution - dual

`Task.readbsolution`

```
readbsolution(filename,compress)
```

Read a binary dump of the task solution.

Parameters
- `filename` (`str`) – A valid file name. (input)
- `compress` (*mosek.compresstype*) – Data compression type. (input)

Groups
Input/Output

`Task.readdata`

```
readdata(filename)
```

Reads an optimization problem and associated data from a file.

Parameters
`filename` (`str`) – A valid file name. (input)

Groups
Input/Output

`Task.readdataformat`

```
readdataformat(filename,format,compress)
```

Reads an optimization problem and associated data from a file.

Parameters
- `filename` (`str`) – A valid file name. (input)
- `format` (*mosek.dataformat*) – File data format. (input)
- `compress` (*mosek.compresstype*) – File compression type. (input)

Groups
Input/Output

`Task.readjsonsol`

```
readjsonsol(filename)
```

Reads a solution file in JSON format (JSOL file) and inserts it in the task. Only the section `Task/solutions` is taken into consideration.

Parameters
`filename` (`str`) – A valid file name. (input)

Groups
Input/Output

`Task.readjsonstring`

```
readjsonstring(data)
```

Load task data from a JSON string, replacing any data that already exists in the task object. All problem data, parameters and other settings are resorted, but if the string contains solutions, the solution status after loading a file is set to unknown, even if it is optimal or otherwise well-defined.

Parameters
`data` (`str`) – Problem data in text format. (input)

Groups
Input/Output

`Task.readlpstring`

```
readlpstring(data)
```

Load task data from a string in LP format, replacing any data that already exists in the task object.

Parameters
`data` (`str`) – Problem data in text format. (input)

Groups
Input/Output

`Task.readopfstring`

```
readopfstring(data)
```

Load task data from a string in OPF format, replacing any data that already exists in the task object.

Parameters
`data` (`str`) – Problem data in text format. (input)

Groups
Input/Output

`Task.readparamfile`

```
readparamfile(filename)
```

Reads **MOSEK** parameters from a file. Data is read from the file `filename` if it is a nonempty string. Otherwise data is read from the file specified by *sparam.param_read_file_name*.

The parameter file must begin with `BEGIN MOSEK` and end with `END MOSEK`; empty lines and lines starting from a `%` sign are ignored; each remaining line contains a valid **MOSEK** parameter name followed by its value (using generic names as in the command-line tool syntax). Example:

```
BEGIN MOSEK
% This is a comment.
MSK_IPAR_PRESOLVE_USE          MSK_OFF
MSK_DPAR_INTPNT_TOL_PFEAS      1.0e-9
END MOSEK
```

Parameters
`filename` (`str`) – A valid file name. (input)

Groups
Input/Output, *Parameters*

`Task.readptfstring`

```
readptfstring(data)
```

Load task data from a PTF string, replacing any data that already exists in the task object. All problem data, parameters and other settings are resorted, but if the string contains solutions, the solution status after loading a file is set to unknown, even if it is optimal or otherwise well-defined.

Parameters
`data` (`str`) – Problem data in text format. (input)

Groups
Input/Output

`Task.readsolution`

```
readsolution(whichsol,filename)
```

Reads a solution file and inserts it as a specified solution in the task. Data is read from the file `filename` if it is a nonempty string. Otherwise data is read from one of the files specified by *sparam.bas_sol_file_name*, *sparam.itr_sol_file_name* or *sparam.int_sol_file_name* depending on which solution is chosen.

Parameters
- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `filename` (`str`) – A valid file name. (input)

Groups
Input/Output

`Task.readsolutionfile`

```
readsolutionfile(filename)
```

Read solution file in format determined by the filename

Parameters
`filename` (`str`) – A valid file name. (input)

Groups
Input/Output

`Task.readsummary`

```
readsummary(whichstream)
```

Prints a short summary of last file that was read.

Parameters
`whichstream` (*mosek.streamtype*) – Index of the stream. (input)

Groups
Input/Output, *Inspecting the task*

`Task.readtask`

```
readtask(filename)
```

Load task data from a file, replacing any data that already exists in the task object. All problem data, parameters and other settings are resorted, but if the file contains solutions, the solution status after loading a file is set to unknown, even if it was optimal or otherwise well-defined when the file was dumped.

See section *The Task Format* for a description of the Task format.

Parameters
`filename` (`str`) – A valid file name. (input)

Groups
Input/Output

`Task.removebarvars`

```
removebarvars(subset)
```

The function removes a subset of the symmetric matrices from the optimization task. This implies that the remaining symmetric matrices are renumbered.

Parameters
`subset` (`int32[]`) – Indexes of symmetric matrices which should be removed. (input)

Groups
Problem data - semidefinite

~~`Task.removecones`~~ *Deprecated*

```
removecones(subset)
```

NOTE: This interface to conic optimization is deprecated and will be removed in a future major release. Conic problems should be specified using the affine conic constraints interface (ACC), see Sec. 6.2 for details.

Removes a number of conic constraints from the problem. This implies that the remaining conic constraints are renumbered. In general, it is much more efficient to remove a cone with a high index than a low index.

Parameters
`subset` (`int32[]`) – Indexes of cones which should be removed. (input)

Groups
Problem data - cones (deprecated)

`Task.removecons`

```
removecons(subset)
```

The function removes a subset of the constraints from the optimization task. This implies that the remaining constraints are renumbered.

Parameters
`subset` (`int32[]`) – Indexes of constraints which should be removed. (input)

Groups
Problem data - constraints, *Problem data - linear part*

`Task.removevars`

```
removevars(subset)
```

The function removes a subset of the variables from the optimization task. This implies that the remaining variables are renumbered.

Parameters
`subset` (`int32[]`) – Indexes of variables which should be removed. (input)

Groups
Problem data - variables, *Problem data - linear part*

`Task.resetdouparam`

```
resetdouparam(param)
```

Resets a double parameter to its default value.

Parameters
`param` (*mosek.dparam*) – Which parameter. (input)

Groups
Parameters

`Task.resetintparam`

```
resetintparam(param)
```

Resets an integer parameter to its default value.

Parameters
`param` (*mosek.iparam*) – Which parameter. (input)

Groups
Parameters

`Task.resetparameters`

```
resetparameters()
```

Resets all the parameters to their default values.

Groups
Parameters

`Task.resetstrparam`

```
resetstrparam(param)
```

Resets a string parameter to its defalt value.

Parameters
`param` (*mosek.sparam*) – Which parameter. (input)

Groups
Parameters

`Task.resizetask`

```
resizetask(maxnumcon,
           maxnumvar,
           maxnumcone,
           maxnumanz,
           maxnumqnz)
```

Sets the amount of preallocated space assigned for each type of data in an optimization task.

It is never mandatory to call this function, since it only gives a hint about the amount of data to preallocate for efficiency reasons.

Please note that the procedure is **destructive** in the sense that all existing data stored in the task is destroyed.

Parameters
- `maxnumcon` (`int32`) – New maximum number of constraints. (input)
- `maxnumvar` (`int32`) – New maximum number of variables. (input)
- `maxnumcone` (`int32`) – New maximum number of cones. (input)
- `maxnumanz` (`int64`) – New maximum number of non-zeros in A. (input)
- `maxnumqnz` (`int64`) – New maximum number of non-zeros in all Q matrices. (input)

Groups
Environment and task management

`Task.sensitivityreport`

```
sensitivityreport(whichstream)
```

Reads a sensitivity format file from a location given by *sparam.sensitivity_file_name* and writes the result to the stream `whichstream`. If *sparam.sensitivity_res_file_name* is set to a non-empty string, then the sensitivity report is also written to a file of this name.

Parameters
`whichstream` (*mosek.streamtype*) – Index of the stream. (input)

Groups
Sensitivity analysis

`Task.set_InfoCallback`

```
set_InfoCallback(callback)
```

Receive callbacks with solver status and information during optimization.

For example:

```
task.set_InfoCallback(lambda code,dinf,iinf,liinf: print("Called from: {0}".
↪format(code)))
```

Parameters
`callback` (*callbackfunc*) – The callback function. (input)

Groups
Callback

`Task.set_Progress`

```
set_Progress(callback)
```

Receive callbacks about current status of the solver during optimization.

For example:

```
task.set_Progress(lambda code: print("Called from: {0}".format(code)))
```

Parameters
`callback` (*progresscallbackfunc*) – The callback function. (input)
Groups
Callback

`Task.set_Stream`

```
set_Stream(whichstream, callback)
```

Directs all output from a task stream to a callback function.

Parameters
- `whichstream` (*streamtype*) – Index of the stream. (input)
- `callback` (*streamfunc*) – The callback function. (input)

Groups
Logging, *Callback*

`Task.solutiondef`

```
solutiondef(whichsol) -> (isdef)
```

Checks whether a solution is defined.

Parameters
`whichsol` (*mosek.soltype*) – Selects a solution. (input)
Return
`isdef` (`bool`) – Is non-zero if the requested solution is defined.
Groups
Solution information

`Task.solutionsummary`

```
solutionsummary(whichstream)
```

Prints a short summary of the current solutions.

Parameters
`whichstream` (*mosek.streamtype*) – Index of the stream. (input)
Groups
Logging, *Solution information*

`Task.solvewithbasis`

```
solvewithbasis(transp,numnz,sub,val) -> (numnzout)
```

If a basic solution is available, then exactly *numcon* basis variables are defined. These *numcon* basis variables are denoted the basis. Associated with the basis is a basis matrix denoted B. This function solves either the linear equation system

$$B\overline{X} = b \tag{15.3}$$

or the system

$$B^T\overline{X} = b \tag{15.4}$$

for the unknowns $\overline{X}$, with b being a user-defined vector. In order to make sense of the solution $\overline{X}$ it is important to know the ordering of the variables in the basis because the ordering specifies how B is constructed. When calling *Task.initbasissolve* an ordering of the basis variables is obtained, which can be used to deduce how **MOSEK** has constructed B. Indeed if the k-th basis variable is variable x_j it implies that

$$B_{i,k} = A_{i,j},\ i = 0, \ldots, \texttt{numcon} - 1.$$

Otherwise if the k-th basis variable is variable x_j^c it implies that

$$B_{i,k} = \begin{cases} -1, & i = j, \\ 0, & i \neq j. \end{cases}$$

The function *Task.initbasissolve* must be called before a call to this function. Please note that this function exploits the sparsity in the vector b to speed up the computations.

Parameters

- `transp` (`bool`) – If this argument is zero, then (15.3) is solved, if non-zero then (15.4) is solved. (input)
- `numnz` (`int32`) – The number of non-zeros in b. (input)
- `sub` (`int32[]`) – As input it contains the positions of non-zeros in b. As output it contains the positions of the non-zeros in $\overline{X}$. It must have room for *numcon* elements. (input/output)
- `val` (`float[]`) – As input it is the vector b as a dense vector (although the positions of non-zeros are specified in `sub` it is required that `val`$[i] = 0$ when $b[i] = 0$). As output `val` is the vector $\overline{X}$ as a dense vector. It must have length *numcon*. (input/output)

Return

`numnzout` (`int32`) – The number of non-zeros in $\overline{X}$.

Groups

Solving systems with basis matrix

~~`Task.strtoconetype`~~ *Deprecated*

```
strtoconetype(str) -> (conetype)
```

NOTE: This interface to conic optimization is deprecated and will be removed in a future major release. Conic problems should be specified using the affine conic constraints interface (ACC), see Sec. 6.2 for details.

Obtains cone type code corresponding to a cone type string.

Parameters

`str` (`str`) – String corresponding to the cone type code `conetype`. (input)

Return

`conetype` (*mosek.conetype*) – The cone type corresponding to the string `str`.

Groups

Names

`Task.strtosk`

```
strtosk(str) -> (sk)
```

Obtains the status key corresponding to an abbreviation string.

Parameters
`str` (`str`) – A status key abbreviation string. (input)

Return
`sk` (*mosek.stakey*) – Status key corresponding to the string.

Groups
Names

~~`Task.toconic`~~ *Deprecated*

```
toconic()
```

This function tries to reformulate a given Quadratically Constrained Quadratic Optimization problem (QCQO) as a Conic Quadratic Optimization problem (CQO). The first step of the reformulation is to convert the quadratic term of the objective function, if any, into a constraint. Then the following steps are repeated for each quadratic constraint:

- a conic constraint is added along with a suitable number of auxiliary variables and constraints;
- the original quadratic constraint is not removed, but all its coefficients are zeroed out.

Note that the reformulation preserves all the original variables.

The conversion is performed in-place, i.e. the task passed as argument is modified on exit. That also means that if the reformulation fails, i.e. the given QCQP is not representable as a CQO, then the task has an undefined state. In some cases, users may want to clone the task to ensure a clean copy is preserved.

Groups
Problem data - quadratic part

`Task.updatesolutioninfo`

```
updatesolutioninfo(whichsol)
```

Update the information items related to the solution.

Parameters
`whichsol` (*mosek.soltype*) – Selects a solution. (input)

Groups
Information items and statistics

`Task.writebsolution`

```
writebsolution(filename,compress)
```

Write a binary dump of the task solution.

Parameters
- `filename` (`str`) – A valid file name. (input)
- `compress` (*mosek.compresstype*) – Data compression type. (input)

Groups
Input/Output

`Task.writedata`

```
writedata(filename)
```

Writes problem data associated with the optimization task to a file in one of the supported formats. See Section *Supported File Formats* for the complete list.

The data file format is determined by the file name extension. To write in compressed format append the extension `.gz`. E.g to write a gzip compressed MPS file use the extension `mps.gz`.

Please note that MPS, LP and OPF files require all variables to have unique names. If a task contains no names, it is possible to write the file with automatically generated anonymous names by setting the *iparam.write_generic_names* parameter to *onoffkey.on*.

Data is written to the file `filename` if it is a nonempty string. Otherwise data is written to the file specified by *sparam.data_file_name*.

Parameters

`filename` (`str`) – A valid file name. (input)

Groups

Input/Output

`Task.writedatastream`

```
writedatastream(format, compress, stream)
```

Writes problem data associated with the optimization task to a stream in one of the supported formats.

The stream should have a `write` method accepting a byte-like object, typically an instance of `io.BytesIO` will be suitable. Examples:

```
    task.writedatastream(dataformat.ptf, compresstype.none, sys.stdout.buffer)
    task.writedatastream(dataformat.task, compresstype.none, open("outfile.task",
↪ "wb"))
```

Parameters

- `format` (*mosek.dataformat*) – Data format. (input)
- `compress` (*mosek.compresstype*) – Selects compression type. (input)
- `stream` (`iostream`) – The output stream. (input)

Groups

Input/Output

`Task.writejsonsol`

```
writejsonsol(filename)
```

Saves the current solutions and solver information items in a JSON file. If the file name has the extensions .gz or .zst, then the file is gzip or Zstd compressed respectively.

Parameters

`filename` (`str`) – A valid file name. (input)

Groups

Input/Output

`Task.writeparamfile`

```
writeparamfile(filename)
```

Writes all the parameters to a parameter file.

Parameters

`filename` (`str`) – A valid file name. (input)

Groups

Input/Output, *Parameters*

`Task.writesolution`

```
writesolution(whichsol,filename)
```

Saves the current basic, interior-point, or integer solution to a file.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `filename` (`str`) – A valid file name. (input)

Groups

Input/Output

`Task.writesolutionfile`

```
writesolutionfile(filename)
```

Write solution file in format determined by the filename

Parameters

`filename` (`str`) – A valid file name. (input)

Groups

Input/Output

`Task.writetask`

```
writetask(filename)
```

Write a binary dump of the task data. This format saves all problem data, coefficients and parameter settings. See section *The Task Format* for a description of the Task format.

Parameters

`filename` (`str`) – A valid file name. (input)

Groups

Input/Output

15.5 Exceptions

`MosekException`

Base exception class for all **MOSEK** exceptions.

`Error`

Exception class used for all error response codes from **MOSEK**.

Implements

MosekException

15.6 Parameters grouped by topic

Analysis

- *dparam.ana_sol_infeas_tol*
- *iparam.ana_sol_basis*
- *iparam.ana_sol_print_violated*
- *iparam.log_ana_pro*

Basis identification

- *dparam.sim_lu_tol_rel_piv*
- *iparam.bi_clean_optimizer*
- *iparam.bi_ignore_max_iter*
- *iparam.bi_ignore_num_error*
- *iparam.bi_max_iterations*
- *iparam.intpnt_basis*
- *iparam.log_bi*
- *iparam.log_bi_freq*

Conic interior-point method

- *dparam.intpnt_co_tol_dfeas*
- *dparam.intpnt_co_tol_infeas*
- *dparam.intpnt_co_tol_mu_red*
- *dparam.intpnt_co_tol_near_rel*
- *dparam.intpnt_co_tol_pfeas*
- *dparam.intpnt_co_tol_rel_gap*

Data check

- *dparam.data_sym_mat_tol*
- *dparam.data_sym_mat_tol_huge*
- *dparam.data_sym_mat_tol_large*
- *dparam.data_tol_aij_huge*
- *dparam.data_tol_aij_large*
- *dparam.data_tol_bound_inf*
- *dparam.data_tol_bound_wrn*
- *dparam.data_tol_c_huge*
- *dparam.data_tol_cj_large*
- *dparam.data_tol_qij*
- *dparam.data_tol_x*
- *dparam.semidefinite_tol_approx*

Data input/output

- *iparam.infeas_report_auto*
- *iparam.log_file*
- *iparam.opf_write_header*
- *iparam.opf_write_hints*
- *iparam.opf_write_line_length*
- *iparam.opf_write_parameters*
- *iparam.opf_write_problem*
- *iparam.opf_write_sol_bas*
- *iparam.opf_write_sol_itg*
- *iparam.opf_write_sol_itr*
- *iparam.opf_write_solutions*
- *iparam.param_read_case_name*
- *iparam.param_read_ign_error*
- *iparam.ptf_write_parameters*
- *iparam.ptf_write_single_psd_terms*
- *iparam.ptf_write_solutions*
- *iparam.ptf_write_transform*
- *iparam.read_async*
- *iparam.read_debug*
- *iparam.read_keep_free_con*
- *iparam.read_mps_format*
- *iparam.read_mps_width*
- *iparam.read_task_ignore_param*
- *iparam.sol_read_name_width*
- *iparam.sol_read_width*
- *iparam.write_async*
- *iparam.write_bas_constraints*
- *iparam.write_bas_head*
- *iparam.write_bas_variables*
- *iparam.write_compression*
- *iparam.write_free_con*
- *iparam.write_generic_names*
- *iparam.write_ignore_incompatible_items*
- *iparam.write_int_constraints*
- *iparam.write_int_head*

- *iparam.write_int_variables*
- *iparam.write_json_indentation*
- *iparam.write_lp_full_obj*
- *iparam.write_lp_line_width*
- *iparam.write_mps_format*
- *iparam.write_mps_int*
- *iparam.write_sol_barvariables*
- *iparam.write_sol_constraints*
- *iparam.write_sol_head*
- *iparam.write_sol_ignore_invalid_names*
- *iparam.write_sol_variables*
- *sparam.bas_sol_file_name*
- *sparam.data_file_name*
- *sparam.debug_file_name*
- *sparam.int_sol_file_name*
- *sparam.itr_sol_file_name*
- *sparam.mio_debug_string*
- *sparam.param_comment_sign*
- *sparam.param_read_file_name*
- *sparam.param_write_file_name*
- *sparam.read_mps_bou_name*
- *sparam.read_mps_obj_name*
- *sparam.read_mps_ran_name*
- *sparam.read_mps_rhs_name*
- *sparam.sensitivity_file_name*
- *sparam.sensitivity_res_file_name*
- *sparam.sol_filter_xc_low*
- *sparam.sol_filter_xc_upr*
- *sparam.sol_filter_xx_low*
- *sparam.sol_filter_xx_upr*
- *sparam.stat_key*
- *sparam.stat_name*

Debugging

- *iparam.auto_sort_a_before_opt*

Dual simplex

- *iparam.sim_dual_crash*
- *iparam.sim_dual_restrict_selection*
- *iparam.sim_dual_selection*

Infeasibility report

- *iparam.infeas_generic_names*
- *iparam.infeas_report_level*
- *iparam.log_infeas_ana*

Interior-point method

- *dparam.intpnt_co_tol_dfeas*
- *dparam.intpnt_co_tol_infeas*
- *dparam.intpnt_co_tol_mu_red*
- *dparam.intpnt_co_tol_near_rel*
- *dparam.intpnt_co_tol_pfeas*
- *dparam.intpnt_co_tol_rel_gap*
- *dparam.intpnt_qo_tol_dfeas*
- *dparam.intpnt_qo_tol_infeas*
- *dparam.intpnt_qo_tol_mu_red*
- *dparam.intpnt_qo_tol_near_rel*
- *dparam.intpnt_qo_tol_pfeas*
- *dparam.intpnt_qo_tol_rel_gap*
- *dparam.intpnt_tol_dfeas*
- *dparam.intpnt_tol_dsafe*
- *dparam.intpnt_tol_infeas*
- *dparam.intpnt_tol_mu_red*
- *dparam.intpnt_tol_path*
- *dparam.intpnt_tol_pfeas*
- *dparam.intpnt_tol_psafe*
- *dparam.intpnt_tol_rel_gap*
- *dparam.intpnt_tol_rel_step*
- *dparam.intpnt_tol_step_size*
- *dparam.qcqo_reformulate_rel_drop_tol*

- *iparam.bi_ignore_max_iter*
- *iparam.bi_ignore_num_error*
- *iparam.intpnt_basis*
- *iparam.intpnt_diff_step*
- *iparam.intpnt_hotstart*
- *iparam.intpnt_max_iterations*
- *iparam.intpnt_max_num_cor*
- *iparam.intpnt_off_col_trh*
- *iparam.intpnt_order_gp_num_seeds*
- *iparam.intpnt_order_method*
- *iparam.intpnt_regularization_use*
- *iparam.intpnt_scaling*
- *iparam.intpnt_solve_form*
- *iparam.intpnt_starting_point*
- *iparam.log_intpnt*

License manager

- *iparam.cache_license*
- *iparam.license_debug*
- *iparam.license_pause_time*
- *iparam.license_suppress_expire_wrns*
- *iparam.license_trh_expiry_wrn*
- *iparam.license_wait*

Logging

- *iparam.heartbeat_sim_freq_ticks*
- *iparam.log*
- *iparam.log_ana_pro*
- *iparam.log_bi*
- *iparam.log_bi_freq*
- *iparam.log_cut_second_opt*
- *iparam.log_expand*
- *iparam.log_feas_repair*
- *iparam.log_file*
- *iparam.log_include_summary*
- *iparam.log_infeas_ana*

- *iparam.log_intpnt*
- *iparam.log_local_info*
- *iparam.log_mio*
- *iparam.log_mio_freq*
- *iparam.log_order*
- *iparam.log_presolve*
- *iparam.log_sensitivity*
- *iparam.log_sensitivity_opt*
- *iparam.log_sim*
- *iparam.log_sim_freq*
- *iparam.log_sim_freq_giga_ticks*
- *iparam.log_storage*

Mixed-integer optimization

- *dparam.mio_clique_table_size_factor*
- *dparam.mio_djc_max_bigm*
- *dparam.mio_max_time*
- *dparam.mio_rel_gap_const*
- *dparam.mio_tol_abs_gap*
- *dparam.mio_tol_abs_relax_int*
- *dparam.mio_tol_feas*
- *dparam.mio_tol_rel_dual_bound_improvement*
- *dparam.mio_tol_rel_gap*
- *iparam.log_mio*
- *iparam.log_mio_freq*
- *iparam.mio_branch_dir*
- *iparam.mio_conflict_analysis_level*
- *iparam.mio_conic_outer_approximation*
- *iparam.mio_construct_sol*
- *iparam.mio_crossover_max_nodes*
- *iparam.mio_cut_clique*
- *iparam.mio_cut_cmir*
- *iparam.mio_cut_gmi*
- *iparam.mio_cut_implied_bound*
- *iparam.mio_cut_knapsack_cover*
- *iparam.mio_cut_lipro*

- *iparam.mio_cut_selection_level*
- *iparam.mio_data_permutation_method*
- *iparam.mio_dual_ray_analysis_level*
- *iparam.mio_feaspump_level*
- *iparam.mio_heuristic_level*
- *iparam.mio_independent_block_level*
- *iparam.mio_max_num_branches*
- *iparam.mio_max_num_relaxs*
- *iparam.mio_max_num_restarts*
- *iparam.mio_max_num_root_cut_rounds*
- *iparam.mio_max_num_solutions*
- *iparam.mio_memory_emphasis_level*
- *iparam.mio_min_rel*
- *iparam.mio_node_optimizer*
- *iparam.mio_node_selection*
- *iparam.mio_numerical_emphasis_level*
- *iparam.mio_opt_face_max_nodes*
- *iparam.mio_perspective_reformulate*
- *iparam.mio_probing_level*
- *iparam.mio_propagate_objective_constraint*
- *iparam.mio_qcqo_reformulation_method*
- *iparam.mio_rens_max_nodes*
- *iparam.mio_rins_max_nodes*
- *iparam.mio_root_optimizer*
- *iparam.mio_seed*
- *iparam.mio_symmetry_level*
- *iparam.mio_var_selection*
- *iparam.mio_vb_detection_level*

Output information

- *iparam.heartbeat_sim_freq_ticks*
- *iparam.infeas_report_level*
- *iparam.license_suppress_expire_wrns*
- *iparam.license_trh_expiry_wrn*
- *iparam.log*
- *iparam.log_bi*
- *iparam.log_bi_freq*
- *iparam.log_cut_second_opt*
- *iparam.log_expand*
- *iparam.log_feas_repair*
- *iparam.log_file*
- *iparam.log_include_summary*
- *iparam.log_infeas_ana*
- *iparam.log_intpnt*
- *iparam.log_local_info*
- *iparam.log_mio*
- *iparam.log_mio_freq*
- *iparam.log_order*
- *iparam.log_sensitivity*
- *iparam.log_sensitivity_opt*
- *iparam.log_sim*
- *iparam.log_sim_freq*
- *iparam.log_sim_freq_giga_ticks*
- *iparam.log_storage*
- *iparam.max_num_warnings*

Overall solver

- *iparam.bi_clean_optimizer*
- *iparam.license_wait*
- *iparam.mio_mode*
- *iparam.optimizer*
- *iparam.presolve_max_num_reductions*
- *iparam.presolve_use*
- *iparam.primal_repair_optimizer*
- *iparam.sensitivity_all*
- *iparam.sensitivity_type*
- *iparam.sim_precision*

Overall system

- *iparam.auto_update_sol_info*
- *iparam.license_wait*
- *iparam.log_storage*
- *iparam.mt_spincount*
- *iparam.num_threads*
- *iparam.remove_unused_solutions*
- *iparam.timing_level*
- *sparam.remote_optserver_host*
- *sparam.remote_tls_cert*
- *sparam.remote_tls_cert_path*

Presolve

- *dparam.folding_tol_eq*
- *dparam.presolve_tol_abs_lindep*
- *dparam.presolve_tol_primal_infeas_perturbation*
- *dparam.presolve_tol_rel_lindep*
- *dparam.presolve_tol_s*
- *dparam.presolve_tol_x*
- *iparam.folding_use*
- *iparam.mio_presolve_aggregator_use*
- *iparam.presolve_eliminator_max_fill*
- *iparam.presolve_eliminator_max_num_tries*
- *iparam.presolve_lindep_abs_work_trh*
- *iparam.presolve_lindep_new*
- *iparam.presolve_lindep_rel_work_trh*
- *iparam.presolve_lindep_use*
- *iparam.presolve_max_num_pass*
- *iparam.presolve_max_num_reductions*
- *iparam.presolve_use*

Primal simplex

- *iparam.sim_primal_crash*
- *iparam.sim_primal_restrict_selection*
- *iparam.sim_primal_selection*

Simplex optimizer

- *dparam.basis_rel_tol_s*
- *dparam.basis_tol_s*
- *dparam.basis_tol_x*
- *dparam.sim_lu_tol_rel_piv*
- *dparam.sim_precision_scaling_extended*
- *dparam.sim_precision_scaling_normal*
- *dparam.simplex_abs_tol_piv*
- *iparam.basis_solve_use_plus_one*
- *iparam.heartbeat_sim_freq_ticks*
- *iparam.log_sim*
- *iparam.log_sim_freq*
- *iparam.log_sim_freq_giga_ticks*
- *iparam.sim_basis_factor_use*
- *iparam.sim_degen*
- *iparam.sim_detect_pwl*
- *iparam.sim_dual_phaseone_method*
- *iparam.sim_exploit_dupvec*
- *iparam.sim_hotstart*
- *iparam.sim_hotstart_lu*
- *iparam.sim_max_iterations*
- *iparam.sim_max_num_setbacks*
- *iparam.sim_non_singular*
- *iparam.sim_precision_boost*
- *iparam.sim_primal_phaseone_method*
- *iparam.sim_refactor_freq*
- *iparam.sim_reformulation*
- *iparam.sim_save_lu*
- *iparam.sim_scaling*
- *iparam.sim_scaling_method*
- *iparam.sim_seed*
- *iparam.sim_solve_form*
- *iparam.sim_switch_optimizer*

Solution input/output

- *iparam.infeas_report_auto*
- *iparam.sol_filter_keep_basic*
- *iparam.sol_read_name_width*
- *iparam.sol_read_width*
- *iparam.write_bas_constraints*
- *iparam.write_bas_head*
- *iparam.write_bas_variables*
- *iparam.write_int_constraints*
- *iparam.write_int_head*
- *iparam.write_int_variables*
- *iparam.write_sol_barvariables*
- *iparam.write_sol_constraints*
- *iparam.write_sol_head*
- *iparam.write_sol_ignore_invalid_names*
- *iparam.write_sol_variables*
- *sparam.bas_sol_file_name*
- *sparam.int_sol_file_name*
- *sparam.itr_sol_file_name*
- *sparam.sol_filter_xc_low*
- *sparam.sol_filter_xc_upr*
- *sparam.sol_filter_xx_low*
- *sparam.sol_filter_xx_upr*

Termination criteria

- *dparam.basis_rel_tol_s*
- *dparam.basis_tol_s*
- *dparam.basis_tol_x*
- *dparam.intpnt_co_tol_dfeas*
- *dparam.intpnt_co_tol_infeas*
- *dparam.intpnt_co_tol_mu_red*
- *dparam.intpnt_co_tol_near_rel*
- *dparam.intpnt_co_tol_pfeas*
- *dparam.intpnt_co_tol_rel_gap*
- *dparam.intpnt_qo_tol_dfeas*
- *dparam.intpnt_qo_tol_infeas*

- *dparam.intpnt_qo_tol_mu_red*
- *dparam.intpnt_qo_tol_near_rel*
- *dparam.intpnt_qo_tol_pfeas*
- *dparam.intpnt_qo_tol_rel_gap*
- *dparam.intpnt_tol_dfeas*
- *dparam.intpnt_tol_infeas*
- *dparam.intpnt_tol_mu_red*
- *dparam.intpnt_tol_pfeas*
- *dparam.intpnt_tol_rel_gap*
- *dparam.lower_obj_cut*
- *dparam.lower_obj_cut_finite_trh*
- *dparam.mio_max_time*
- *dparam.mio_rel_gap_const*
- *dparam.mio_tol_rel_gap*
- *dparam.optimizer_max_ticks*
- *dparam.optimizer_max_time*
- *dparam.sim_precision_scaling_extended*
- *dparam.sim_precision_scaling_normal*
- *dparam.upper_obj_cut*
- *dparam.upper_obj_cut_finite_trh*
- *iparam.bi_max_iterations*
- *iparam.intpnt_max_iterations*
- *iparam.mio_max_num_branches*
- *iparam.mio_max_num_root_cut_rounds*
- *iparam.mio_max_num_solutions*
- *iparam.sim_max_iterations*

Other

- *iparam.compress_statfile*
- *iparam.getdual_convert_lmis*
- *iparam.ng*
- *iparam.remote_use_compression*

15.7 Parameters (alphabetical list sorted by type)

- *Double parameters*
- *Integer parameters*
- *String parameters*

15.7.1 Double parameters

`dparam`

The enumeration type containing all double parameters.

`dparam.ana_sol_infeas_tol`

If a constraint violates its bound with an amount larger than this value, the constraint name, index and violation will be printed by the solution analyzer.

Default
1e-6

Accepted
[0.0; +inf]

Example
`task.putdouparam(dparam.ana_sol_infeas_tol, 1e-6)`

Generic name
MSK_DPAR_ANA_SOL_INFEAS_TOL

Groups
Analysis

`dparam.basis_rel_tol_s`

Maximum relative dual bound violation allowed in an optimal basic solution.

Default
1.0e-12

Accepted
[0.0; +inf]

Example
`task.putdouparam(dparam.basis_rel_tol_s, 1.0e-12)`

Generic name
MSK_DPAR_BASIS_REL_TOL_S

Groups
Simplex optimizer, *Termination criteria*

`dparam.basis_tol_s`

Maximum absolute dual bound violation in an optimal basic solution.

Default
1.0e-6

Accepted
[1.0e-9; +inf]

Example
`task.putdouparam(dparam.basis_tol_s, 1.0e-6)`

Generic name
MSK_DPAR_BASIS_TOL_S

Groups
Simplex optimizer, *Termination criteria*

`dparam.basis_tol_x`

Maximum absolute primal bound violation allowed in an optimal basic solution.

Default
1.0e-6
Accepted
[1.0e-9; +inf]
Example
`task.putdouparam(dparam.basis_tol_x, 1.0e-6)`
Generic name
MSK_DPAR_BASIS_TOL_X
Groups
Simplex optimizer, *Termination criteria*

`dparam.data_sym_mat_tol`

Absolute zero tolerance for elements in in symmetric matrices. If any value in a symmetric matrix is smaller than this parameter in absolute terms **MOSEK** will treat the values as zero and generate a warning.

Default
1.0e-12
Accepted
[1.0e-16; 1.0e-6]
Example
`task.putdouparam(dparam.data_sym_mat_tol, 1.0e-12)`
Generic name
MSK_DPAR_DATA_SYM_MAT_TOL
Groups
Data check

`dparam.data_sym_mat_tol_huge`

An element in a symmetric matrix which is larger than this value in absolute size causes an error.

Default
1.0e20
Accepted
[0.0; +inf]
Example
`task.putdouparam(dparam.data_sym_mat_tol_huge, 1.0e20)`
Generic name
MSK_DPAR_DATA_SYM_MAT_TOL_HUGE
Groups
Data check

`dparam.data_sym_mat_tol_large`

An element in a symmetric matrix which is larger than this value in absolute size causes a warning message to be printed.

Default
1.0e10
Accepted
[0.0; +inf]
Example
`task.putdouparam(dparam.data_sym_mat_tol_large, 1.0e10)`
Generic name
MSK_DPAR_DATA_SYM_MAT_TOL_LARGE
Groups
Data check

`dparam.data_tol_aij_huge`

An element in A which is larger than this value in absolute size causes an error.

Default
1.0e20

Accepted
[0.0; +inf]

Example
```
task.putdouparam(dparam.data_tol_aij_huge, 1.0e20)
```

Generic name
MSK_DPAR_DATA_TOL_AIJ_HUGE

Groups
Data check

`dparam.data_tol_aij_large`

An element in A which is larger than this value in absolute size causes a warning message to be printed.

Default
1.0e10

Accepted
[0.0; +inf]

Example
```
task.putdouparam(dparam.data_tol_aij_large, 1.0e10)
```

Generic name
MSK_DPAR_DATA_TOL_AIJ_LARGE

Groups
Data check

`dparam.data_tol_bound_inf`

Any bound which in absolute value is greater than this parameter is considered infinite.

Default
1.0e16

Accepted
[0.0; +inf]

Example
```
task.putdouparam(dparam.data_tol_bound_inf, 1.0e16)
```

Generic name
MSK_DPAR_DATA_TOL_BOUND_INF

Groups
Data check

`dparam.data_tol_bound_wrn`

If a bound value is larger than this value in absolute size, then a warning message is issued.

Default
1.0e8

Accepted
[0.0; +inf]

Example
```
task.putdouparam(dparam.data_tol_bound_wrn, 1.0e8)
```

Generic name
MSK_DPAR_DATA_TOL_BOUND_WRN

Groups
Data check

`dparam.data_tol_c_huge`

An element in c which is larger than the value of this parameter in absolute terms is considered to be huge and generates an error.

Default
1.0e16
Accepted
[0.0; +inf]
Example
`task.putdouparam(dparam.data_tol_c_huge, 1.0e16)`
Generic name
MSK_DPAR_DATA_TOL_C_HUGE
Groups
Data check

`dparam.data_tol_cj_large`

An element in c which is larger than this value in absolute terms causes a warning message to be printed.

Default
1.0e8
Accepted
[0.0; +inf]
Example
`task.putdouparam(dparam.data_tol_cj_large, 1.0e8)`
Generic name
MSK_DPAR_DATA_TOL_CJ_LARGE
Groups
Data check

`dparam.data_tol_qij`

Absolute zero tolerance for elements in Q matrices.

Default
1.0e-16
Accepted
[0.0; +inf]
Example
`task.putdouparam(dparam.data_tol_qij, 1.0e-16)`
Generic name
MSK_DPAR_DATA_TOL_QIJ
Groups
Data check

`dparam.data_tol_x`

Zero tolerance for constraints and variables i.e. if the distance between the lower and upper bound is less than this value, then the lower and upper bound is considered identical.

Default
1.0e-8
Accepted
[0.0; +inf]
Example
`task.putdouparam(dparam.data_tol_x, 1.0e-8)`
Generic name
MSK_DPAR_DATA_TOL_X
Groups
Data check

`dparam.folding_tol_eq`

Tolerance for coefficient equality during folding.

Default
1e-9

Accepted
[0.0; +inf]

Example
```
task.putdouparam(dparam.folding_tol_eq, 1e-9)
```

Generic name
MSK_DPAR_FOLDING_TOL_EQ

Groups
Presolve

`dparam.intpnt_co_tol_dfeas`

Dual feasibility tolerance used by the interior-point optimizer for conic problems.

Default
1.0e-8

Accepted
[0.0; 1.0]

Example
```
task.putdouparam(dparam.intpnt_co_tol_dfeas, 1.0e-8)
```

See also
dparam.intpnt_co_tol_near_rel

Generic name
MSK_DPAR_INTPNT_CO_TOL_DFEAS

Groups
Interior-point method, *Termination criteria*, *Conic interior-point method*

`dparam.intpnt_co_tol_infeas`

Infeasibility tolerance used by the interior-point optimizer for conic problems. Controls when the interior-point optimizer declares the model primal or dual infeasible. A small number means the optimizer gets more conservative about declaring the model infeasible.

Default
1.0e-12

Accepted
[0.0; 1.0]

Example
```
task.putdouparam(dparam.intpnt_co_tol_infeas, 1.0e-12)
```

Generic name
MSK_DPAR_INTPNT_CO_TOL_INFEAS

Groups
Interior-point method, *Termination criteria*, *Conic interior-point method*

`dparam.intpnt_co_tol_mu_red`

Relative complementarity gap tolerance used by the interior-point optimizer for conic problems.

Default
1.0e-8

Accepted
[0.0; 1.0]

Example
```
task.putdouparam(dparam.intpnt_co_tol_mu_red, 1.0e-8)
```

Generic name
MSK_DPAR_INTPNT_CO_TOL_MU_RED

Groups
Interior-point method, *Termination criteria*, *Conic interior-point method*

`dparam.intpnt_co_tol_near_rel`

Optimality tolerance used by the interior-point optimizer for conic problems. If **MOSEK** cannot compute a solution that has the prescribed accuracy then it will check if the solution found satisfies the termination criteria with all tolerances multiplied by the value of this parameter. If yes, then the solution is also declared optimal.

Default
1000
Accepted
[1.0; +inf]
Example
`task.putdouparam(dparam.intpnt_co_tol_near_rel, 1000)`
Generic name
MSK_DPAR_INTPNT_CO_TOL_NEAR_REL
Groups
Interior-point method, *Termination criteria*, *Conic interior-point method*

`dparam.intpnt_co_tol_pfeas`

Primal feasibility tolerance used by the interior-point optimizer for conic problems.

Default
1.0e-8
Accepted
[0.0; 1.0]
Example
`task.putdouparam(dparam.intpnt_co_tol_pfeas, 1.0e-8)`
See also
dparam.intpnt_co_tol_near_rel
Generic name
MSK_DPAR_INTPNT_CO_TOL_PFEAS
Groups
Interior-point method, *Termination criteria*, *Conic interior-point method*

`dparam.intpnt_co_tol_rel_gap`

Relative gap termination tolerance used by the interior-point optimizer for conic problems.

Default
1.0e-8
Accepted
[0.0; 1.0]
Example
`task.putdouparam(dparam.intpnt_co_tol_rel_gap, 1.0e-8)`
See also
dparam.intpnt_co_tol_near_rel
Generic name
MSK_DPAR_INTPNT_CO_TOL_REL_GAP
Groups
Interior-point method, *Termination criteria*, *Conic interior-point method*

`dparam.intpnt_qo_tol_dfeas`

Dual feasibility tolerance used by the interior-point optimizer for quadratic problems.

Default
1.0e-8

Accepted
[0.0; 1.0]
Example
`task.putdouparam(dparam.intpnt_qo_tol_dfeas, 1.0e-8)`
See also
dparam.intpnt_qo_tol_near_rel
Generic name
MSK_DPAR_INTPNT_QO_TOL_DFEAS
Groups
Interior-point method, *Termination criteria*

`dparam.intpnt_qo_tol_infeas`

Infeasibility tolerance used by the interior-point optimizer for quadratic problems. Controls when the interior-point optimizer declares the model primal or dual infeasible. A small number means the optimizer gets more conservative about declaring the model infeasible.

Default
1.0e-12
Accepted
[0.0; 1.0]
Example
`task.putdouparam(dparam.intpnt_qo_tol_infeas, 1.0e-12)`
Generic name
MSK_DPAR_INTPNT_QO_TOL_INFEAS
Groups
Interior-point method, *Termination criteria*

`dparam.intpnt_qo_tol_mu_red`

Relative complementarity gap tolerance used by the interior-point optimizer for quadratic problems.

Default
1.0e-8
Accepted
[0.0; 1.0]
Example
`task.putdouparam(dparam.intpnt_qo_tol_mu_red, 1.0e-8)`
Generic name
MSK_DPAR_INTPNT_QO_TOL_MU_RED
Groups
Interior-point method, *Termination criteria*

`dparam.intpnt_qo_tol_near_rel`

Optimality tolerance used by the interior-point optimizer for quadratic problems. If **MOSEK** cannot compute a solution that has the prescribed accuracy then it will check if the solution found satisfies the termination criteria with all tolerances multiplied by the value of this parameter. If yes, then the solution is also declared optimal.

Default
1000
Accepted
[1.0; +inf]
Example
`task.putdouparam(dparam.intpnt_qo_tol_near_rel, 1000)`
Generic name
MSK_DPAR_INTPNT_QO_TOL_NEAR_REL
Groups
Interior-point method, *Termination criteria*

`dparam.intpnt_qo_tol_pfeas`

Primal feasibility tolerance used by the interior-point optimizer for quadratic problems.

Default
1.0e-8

Accepted
[0.0; 1.0]

Example
```
task.putdouparam(dparam.intpnt_qo_tol_pfeas, 1.0e-8)
```

See also
dparam.intpnt_qo_tol_near_rel

Generic name
MSK_DPAR_INTPNT_QO_TOL_PFEAS

Groups
Interior-point method, *Termination criteria*

`dparam.intpnt_qo_tol_rel_gap`

Relative gap termination tolerance used by the interior-point optimizer for quadratic problems.

Default
1.0e-8

Accepted
[0.0; 1.0]

Example
```
task.putdouparam(dparam.intpnt_qo_tol_rel_gap, 1.0e-8)
```

See also
dparam.intpnt_qo_tol_near_rel

Generic name
MSK_DPAR_INTPNT_QO_TOL_REL_GAP

Groups
Interior-point method, *Termination criteria*

`dparam.intpnt_tol_dfeas`

Dual feasibility tolerance used by the interior-point optimizer for linear problems.

Default
1.0e-8

Accepted
[0.0; 1.0]

Example
```
task.putdouparam(dparam.intpnt_tol_dfeas, 1.0e-8)
```

Generic name
MSK_DPAR_INTPNT_TOL_DFEAS

Groups
Interior-point method, *Termination criteria*

`dparam.intpnt_tol_dsafe`

Controls the initial dual starting point used by the interior-point optimizer. If the interior-point optimizer converges slowly and/or the constraint or variable bounds are very large, then it might be worthwhile to increase this value.

Default
1.0

Accepted
[1.0e-4; +inf]

Example
```
task.putdouparam(dparam.intpnt_tol_dsafe, 1.0)
```

Generic name
MSK_DPAR_INTPNT_TOL_DSAFE

Groups
Interior-point method

`dparam.intpnt_tol_infeas`

Infeasibility tolerance used by the interior-point optimizer for linear problems. Controls when the interior-point optimizer declares the model primal or dual infeasible. A small number means the optimizer gets more conservative about declaring the model infeasible.

Default
1.0e-10

Accepted
[0.0; 1.0]

Example
```
task.putdouparam(dparam.intpnt_tol_infeas, 1.0e-10)
```

Generic name
MSK_DPAR_INTPNT_TOL_INFEAS

Groups
Interior-point method, *Termination criteria*

`dparam.intpnt_tol_mu_red`

Relative complementarity gap tolerance used by the interior-point optimizer for linear problems.

Default
1.0e-16

Accepted
[0.0; 1.0]

Example
```
task.putdouparam(dparam.intpnt_tol_mu_red, 1.0e-16)
```

Generic name
MSK_DPAR_INTPNT_TOL_MU_RED

Groups
Interior-point method, *Termination criteria*

`dparam.intpnt_tol_path`

Controls how close the interior-point optimizer follows the central path. A large value of this parameter means the central path is followed very closely. On numerically unstable problems it may be worthwhile to increase this parameter.

Default
1.0e-8

Accepted
[0.0; 0.9999]

Example
```
task.putdouparam(dparam.intpnt_tol_path, 1.0e-8)
```

Generic name
MSK_DPAR_INTPNT_TOL_PATH

Groups
Interior-point method

`dparam.intpnt_tol_pfeas`

Primal feasibility tolerance used by the interior-point optimizer for linear problems.

Default
1.0e-8

Accepted
[0.0; 1.0]

Example
`task.putdouparam(dparam.intpnt_tol_pfeas, 1.0e-8)`
Generic name
MSK_DPAR_INTPNT_TOL_PFEAS
Groups
Interior-point method, *Termination criteria*

`dparam.intpnt_tol_psafe`

Controls the initial primal starting point used by the interior-point optimizer. If the interior-point optimizer converges slowly and/or the constraint or variable bounds are very large, then it may be worthwhile to increase this value.

Default
1.0
Accepted
[1.0e-4; +inf]
Example
`task.putdouparam(dparam.intpnt_tol_psafe, 1.0)`
Generic name
MSK_DPAR_INTPNT_TOL_PSAFE
Groups
Interior-point method

`dparam.intpnt_tol_rel_gap`

Relative gap termination tolerance used by the interior-point optimizer for linear problems.

Default
1.0e-8
Accepted
[1.0e-14; +inf]
Example
`task.putdouparam(dparam.intpnt_tol_rel_gap, 1.0e-8)`
Generic name
MSK_DPAR_INTPNT_TOL_REL_GAP
Groups
Termination criteria, *Interior-point method*

`dparam.intpnt_tol_rel_step`

Relative step size to the boundary for linear and quadratic optimization problems.

Default
0.9999
Accepted
[1.0e-4; 0.999999]
Example
`task.putdouparam(dparam.intpnt_tol_rel_step, 0.9999)`
Generic name
MSK_DPAR_INTPNT_TOL_REL_STEP
Groups
Interior-point method

`dparam.intpnt_tol_step_size`

Minimal step size tolerance. If the step size falls below the value of this parameter, then the interior-point optimizer assumes that it is stalled. In other words the interior-point optimizer does not make any progress and therefore it is better to stop.

Default
1.0e-6

Accepted
[0.0; 1.0]

Example
`task.putdouparam(dparam.intpnt_tol_step_size, 1.0e-6)`

Generic name
MSK_DPAR_INTPNT_TOL_STEP_SIZE

Groups
Interior-point method

`dparam.lower_obj_cut`

If either a primal or dual feasible solution is found proving that the optimal objective value is outside the interval [*dparam.lower_obj_cut*, *dparam.upper_obj_cut*], then **MOSEK** is terminated.

Default
-INFINITY

Accepted
[-inf; +inf]

Example
`task.putdouparam(dparam.lower_obj_cut, -INFINITY)`

See also
dparam.lower_obj_cut_finite_trh

Generic name
MSK_DPAR_LOWER_OBJ_CUT

Groups
Termination criteria

`dparam.lower_obj_cut_finite_trh`

If the lower objective cut is less than the value of this parameter value, then the lower objective cut i.e. *dparam.lower_obj_cut* is treated as $-\infty$.

Default
-0.5e30

Accepted
[-inf; +inf]

Example
`task.putdouparam(dparam.lower_obj_cut_finite_trh, -0.5e30)`

Generic name
MSK_DPAR_LOWER_OBJ_CUT_FINITE_TRH

Groups
Termination criteria

`dparam.mio_clique_table_size_factor`

Controlls the maximum size of the clique table as a factor of the number of nonzeros in the A matrix. A negative value implies **MOSEK** decides.

Default
-1

Accepted
[-1; +inf]

Example
`task.putdouparam(dparam.mio_clique_table_size_factor, -1)`

Generic name
MSK_DPAR_MIO_CLIQUE_TABLE_SIZE_FACTOR

Groups
Mixed-integer optimization

`dparam.mio_djc_max_bigm`

Maximum allowed big-M value when reformulating disjunctive constraints to linear constraints. Higher values make it more likely that a disjunction is reformulated to linear constraints, but also increase the risk of numerical problems.

Default
1.0e6

Accepted
[0; +inf]

Example
```
task.putdouparam(dparam.mio_djc_max_bigm, 1.0e6)
```

Generic name
MSK_DPAR_MIO_DJC_MAX_BIGM

Groups
Mixed-integer optimization

`dparam.mio_max_time`

This parameter limits the maximum time spent by the mixed-integer optimizer (in seconds). A negative number means infinity.

Default
-1.0

Accepted
[-inf; +inf]

Example
```
task.putdouparam(dparam.mio_max_time, -1.0)
```

Generic name
MSK_DPAR_MIO_MAX_TIME

Groups
Mixed-integer optimization, *Termination criteria*

`dparam.mio_rel_gap_const`

This value is used to compute the relative gap for the solution to a mixed-integer optimization problem.

Default
1.0e-10

Accepted
[1.0e-15; +inf]

Example
```
task.putdouparam(dparam.mio_rel_gap_const, 1.0e-10)
```

Generic name
MSK_DPAR_MIO_REL_GAP_CONST

Groups
Mixed-integer optimization, *Termination criteria*

`dparam.mio_tol_abs_gap`

Absolute optimality tolerance employed by the mixed-integer optimizer.

Default
0.0

Accepted
[0.0; +inf]

Example
```
task.putdouparam(dparam.mio_tol_abs_gap, 0.0)
```

Generic name
MSK_DPAR_MIO_TOL_ABS_GAP

Groups
Mixed-integer optimization

`dparam.mio_tol_abs_relax_int`

Absolute integer feasibility tolerance. If the distance to the nearest integer is less than this tolerance then an integer constraint is assumed to be satisfied.

Default
1.0e-5
Accepted
[1e-9; +inf]
Example
`task.putdouparam(dparam.mio_tol_abs_relax_int, 1.0e-5)`
Generic name
MSK_DPAR_MIO_TOL_ABS_RELAX_INT
Groups
Mixed-integer optimization

`dparam.mio_tol_feas`

Feasibility tolerance for mixed integer solver.

Default
1.0e-6
Accepted
[1e-9; 1e-3]
Example
`task.putdouparam(dparam.mio_tol_feas, 1.0e-6)`
Generic name
MSK_DPAR_MIO_TOL_FEAS
Groups
Mixed-integer optimization

`dparam.mio_tol_rel_dual_bound_improvement`

If the relative improvement of the dual bound is smaller than this value, the solver will terminate the root cut generation. A value of 0.0 means that the value is selected automatically.

Default
0.0
Accepted
[0.0; 1.0]
Example
`task.putdouparam(dparam.mio_tol_rel_dual_bound_improvement, 0.0)`
Generic name
MSK_DPAR_MIO_TOL_REL_DUAL_BOUND_IMPROVEMENT
Groups
Mixed-integer optimization

`dparam.mio_tol_rel_gap`

Relative optimality tolerance employed by the mixed-integer optimizer.

Default
1.0e-4
Accepted
[0.0; +inf]
Example
`task.putdouparam(dparam.mio_tol_rel_gap, 1.0e-4)`
Generic name
MSK_DPAR_MIO_TOL_REL_GAP

Groups
Mixed-integer optimization, *Termination criteria*

`dparam.optimizer_max_ticks`

CURRENTLY NOT IN USE.

Maximum amount of ticks the optimizer is allowed to spent on the optimization. A negative number means infinity.

Default
-1.0
Accepted
[-inf; +inf]
Example
`task.putdouparam(dparam.optimizer_max_ticks, -1.0)`
Generic name
MSK_DPAR_OPTIMIZER_MAX_TICKS
Groups
Termination criteria

`dparam.optimizer_max_time`

Maximum amount of time the optimizer is allowed to spent on the optimization (in seconds). A negative number means infinity.

Default
-1.0
Accepted
[-inf; +inf]
Example
`task.putdouparam(dparam.optimizer_max_time, -1.0)`
Generic name
MSK_DPAR_OPTIMIZER_MAX_TIME
Groups
Termination criteria

`dparam.presolve_tol_abs_lindep`

Absolute tolerance employed by the linear dependency checker.

Default
1.0e-6
Accepted
[0.0; +inf]
Example
`task.putdouparam(dparam.presolve_tol_abs_lindep, 1.0e-6)`
Generic name
MSK_DPAR_PRESOLVE_TOL_ABS_LINDEP
Groups
Presolve

`dparam.presolve_tol_primal_infeas_perturbation`

The presolve is allowed to perturb a bound on a constraint or variable by this amount if it removes an infeasibility.

Default
1.0e-6
Accepted
[0.0; +inf]

Example
`task.putdouparam(dparam.presolve_tol_primal_infeas_perturbation, 1.0e-6)`

Generic name
MSK_DPAR_PRESOLVE_TOL_PRIMAL_INFEAS_PERTURBATION

Groups
Presolve

`dparam.presolve_tol_rel_lindep`

Relative tolerance employed by the linear dependency checker.

Default
1.0e-10

Accepted
[0.0; +inf]

Example
`task.putdouparam(dparam.presolve_tol_rel_lindep, 1.0e-10)`

Generic name
MSK_DPAR_PRESOLVE_TOL_REL_LINDEP

Groups
Presolve

`dparam.presolve_tol_s`

Absolute zero tolerance employed for s_i in the presolve.

Default
1.0e-8

Accepted
[0.0; +inf]

Example
`task.putdouparam(dparam.presolve_tol_s, 1.0e-8)`

Generic name
MSK_DPAR_PRESOLVE_TOL_S

Groups
Presolve

`dparam.presolve_tol_x`

Absolute zero tolerance employed for x_j in the presolve.

Default
1.0e-8

Accepted
[0.0; +inf]

Example
`task.putdouparam(dparam.presolve_tol_x, 1.0e-8)`

Generic name
MSK_DPAR_PRESOLVE_TOL_X

Groups
Presolve

`dparam.qcqo_reformulate_rel_drop_tol`

This parameter determines when columns are dropped in incomplete Cholesky factorization during reformulation of quadratic problems.

Default
1e-15

Accepted
[0; +inf]

Example
`task.putdouparam(dparam.qcqo_reformulate_rel_drop_tol, 1e-15)`
Generic name
MSK_DPAR_QCQO_REFORMULATE_REL_DROP_TOL
Groups
Interior-point method

`dparam.semidefinite_tol_approx`

Tolerance to define a matrix to be positive semidefinite.

Default
1.0e-10
Accepted
[1.0e-15; +inf]
Example
`task.putdouparam(dparam.semidefinite_tol_approx, 1.0e-10)`
Generic name
MSK_DPAR_SEMIDEFINITE_TOL_APPROX
Groups
Data check

`dparam.sim_lu_tol_rel_piv`

Relative pivot tolerance employed when computing the LU factorization of the basis in the simplex optimizers and in the basis identification procedure. A value closer to 1.0 generally improves numerical stability but typically also implies an increase in the computational work.

Default
0.01
Accepted
[1.0e-6; 0.999999]
Example
`task.putdouparam(dparam.sim_lu_tol_rel_piv, 0.01)`
Generic name
MSK_DPAR_SIM_LU_TOL_REL_PIV
Groups
Basis identification, *Simplex optimizer*

`dparam.sim_precision_scaling_extended`

Experimental. Usage not recommended.

Default
2.0
Accepted
[1.0; +inf]
Example
`task.putdouparam(dparam.sim_precision_scaling_extended, 2.0)`
Generic name
MSK_DPAR_SIM_PRECISION_SCALING_EXTENDED
Groups
Simplex optimizer, *Termination criteria*

`dparam.sim_precision_scaling_normal`

Experimental. Usage not recommended.

Default
1.0
Accepted
[1.0; +inf]

Example
`task.putdouparam(dparam.sim_precision_scaling_normal, 1.0)`
Generic name
MSK_DPAR_SIM_PRECISION_SCALING_NORMAL
Groups
Simplex optimizer, *Termination criteria*

`dparam.simplex_abs_tol_piv`

Absolute pivot tolerance employed by the simplex optimizers.

Default
1.0e-7
Accepted
[1.0e-12; +inf]
Example
`task.putdouparam(dparam.simplex_abs_tol_piv, 1.0e-7)`
Generic name
MSK_DPAR_SIMPLEX_ABS_TOL_PIV
Groups
Simplex optimizer

`dparam.upper_obj_cut`

If either a primal or dual feasible solution is found proving that the optimal objective value is outside the interval [*dparam.lower_obj_cut*, *dparam.upper_obj_cut*], then **MOSEK** is terminated.

Default
INFINITY
Accepted
[-inf; +inf]
Example
`task.putdouparam(dparam.upper_obj_cut, INFINITY)`
See also
dparam.upper_obj_cut_finite_trh
Generic name
MSK_DPAR_UPPER_OBJ_CUT
Groups
Termination criteria

`dparam.upper_obj_cut_finite_trh`

If the upper objective cut is greater than the value of this parameter, then the upper objective cut *dparam.upper_obj_cut* is treated as ∞.

Default
0.5e30
Accepted
[-inf; +inf]
Example
`task.putdouparam(dparam.upper_obj_cut_finite_trh, 0.5e30)`
Generic name
MSK_DPAR_UPPER_OBJ_CUT_FINITE_TRH
Groups
Termination criteria

15.7.2 Integer parameters

`iparam`

The enumeration type containing all integer parameters.

`iparam.ana_sol_basis`

Controls whether the basis matrix is analyzed in solution analyzer.

Default
on

Accepted
on, *off* (see *onoffkey*)

Example
```
task.putintparam(iparam.ana_sol_basis, onoffkey.on)
```

Generic name
MSK_IPAR_ANA_SOL_BASIS

Groups
Analysis

`iparam.ana_sol_print_violated`

A parameter of the problem analyzer. Controls whether a list of violated constraints is printed. All constraints violated by more than the value set by the parameter *dparam.ana_sol_infeas_tol* will be printed.

Default
off

Accepted
on, *off* (see *onoffkey*)

Example
```
task.putintparam(iparam.ana_sol_print_violated, onoffkey.off)
```

Generic name
MSK_IPAR_ANA_SOL_PRINT_VIOLATED

Groups
Analysis

`iparam.auto_sort_a_before_opt`

Controls whether the elements in each column of A are sorted before an optimization is performed. This is not required but makes the optimization more deterministic.

Default
off

Accepted
on, *off* (see *onoffkey*)

Example
```
task.putintparam(iparam.auto_sort_a_before_opt, onoffkey.off)
```

Generic name
MSK_IPAR_AUTO_SORT_A_BEFORE_OPT

Groups
Debugging

`iparam.auto_update_sol_info`

Controls whether the solution information items are automatically updated after an optimization is performed.

Default
off

Accepted
on, *off* (see *onoffkey*)

Example
`task.putintparam(iparam.auto_update_sol_info, onoffkey.off)`

Generic name
MSK_IPAR_AUTO_UPDATE_SOL_INFO

Groups
Overall system

`iparam.basis_solve_use_plus_one`

If a slack variable is in the basis, then the corresponding column in the basis is a unit vector with -1 in the right position. However, if this parameter is set to *onoffkey.on*, -1 is replaced by 1.

This has significance for the results returned by the *Task.solvewithbasis* function.

Default
off

Accepted
on, *off* (see *onoffkey*)

Example
`task.putintparam(iparam.basis_solve_use_plus_one, onoffkey.off)`

Generic name
MSK_IPAR_BASIS_SOLVE_USE_PLUS_ONE

Groups
Simplex optimizer

`iparam.bi_clean_optimizer`

Controls which simplex optimizer is used in the clean-up phase. Anything else than *optimizertype.primal_simplex* or *optimizertype.dual_simplex* is equivalent to *optimizertype.free_simplex*.

Default
free

Accepted
free, *intpnt*, *conic*, *primal_simplex*, *dual_simplex*, *new_primal_simplex*, *new_dual_simplex*, *free_simplex*, *mixed_int* (see *optimizertype*)

Example
`task.putintparam(iparam.bi_clean_optimizer, optimizertype.free)`

Generic name
MSK_IPAR_BI_CLEAN_OPTIMIZER

Groups
Basis identification, *Overall solver*

`iparam.bi_ignore_max_iter`

If the parameter *iparam.intpnt_basis* has the value *basindtype.no_error* and the interior-point optimizer has terminated due to maximum number of iterations, then basis identification is performed if this parameter has the value *onoffkey.on*.

Default
off

Accepted
on, *off* (see *onoffkey*)

Example
`task.putintparam(iparam.bi_ignore_max_iter, onoffkey.off)`

Generic name
MSK_IPAR_BI_IGNORE_MAX_ITER

Groups
Interior-point method, *Basis identification*

`iparam.bi_ignore_num_error`

If the parameter *iparam.intpnt_basis* has the value *basindtype.no_error* and the interior-point optimizer has terminated due to a numerical problem, then basis identification is performed if this parameter has the value *onoffkey.on*.

Default
off

Accepted
on, *off* (see *onoffkey*)

Example
```
task.putintparam(iparam.bi_ignore_num_error, onoffkey.off)
```

Generic name
MSK_IPAR_BI_IGNORE_NUM_ERROR

Groups
Interior-point method, *Basis identification*

`iparam.bi_max_iterations`

Controls the maximum number of simplex iterations allowed to optimize a basis after the basis identification.

Default
1000000

Accepted
[0; +inf]

Example
```
task.putintparam(iparam.bi_max_iterations, 1000000)
```

Generic name
MSK_IPAR_BI_MAX_ITERATIONS

Groups
Basis identification, *Termination criteria*

`iparam.cache_license`

Specifies if the license is kept checked out for the lifetime of the **MOSEK** environment/model/process (*onoffkey.on*) or returned to the server immediately after the optimization (*onoffkey.off*).

By default the license is checked out for the lifetime of the **MOSEK** environment by the first call to *Task.optimize*.

Check-in and check-out of licenses have an overhead. Frequent communication with the license server should be avoided.

Default
on

Accepted
on, *off* (see *onoffkey*)

Example
```
task.putintparam(iparam.cache_license, onoffkey.on)
```

Generic name
MSK_IPAR_CACHE_LICENSE

Groups
License manager

`iparam.compress_statfile`

Control compression of stat files.

Default
on

Accepted
on, *off* (see *onoffkey*)

Example
```
task.putintparam(iparam.compress_statfile, onoffkey.on)
```

Generic name
MSK_IPAR_COMPRESS_STATFILE

`iparam.folding_use`

Controls whether and how to use problem folding (symmetry detection for continuous problems). Note that for symmetry detection for mixed-integer problems one should instead use the parameter *iparam.mio_symmetry_level*.

Default
free_unless_basic

Accepted
off, *free*, *free_unless_basic*, *force* (see *foldingmode*)

Example
```
task.putintparam(iparam.folding_use, foldingmode.free_unless_basic)
```

Generic name
MSK_IPAR_FOLDING_USE

Groups
Presolve

`iparam.getdual_convert_lmis`

Whether to perform LMI detection and optimization in the user-level dualizer.

Default
on

Accepted
on, *off* (see *onoffkey*)

Example
```
task.putintparam(iparam.getdual_convert_lmis, onoffkey.on)
```

Generic name
MSK_IPAR_GETDUAL_CONVERT_LMIS

`iparam.heartbeat_sim_freq_ticks`

Controls how frequent the new simplex optimizer calls the user-defined callback function is called.

- -1. Logging is disabled.
- 0. Logging at highest frequency (every iteration).
- ≥ 1. Logging at given frequency measured in ticks.

Default
1000000

Accepted
[-1; +inf]

Example
```
task.putintparam(iparam.heartbeat_sim_freq_ticks, 1000000)
```

Generic name
MSK_IPAR_HEARTBEAT_SIM_FREQ_TICKS

Groups
Simplex optimizer, *Output information*, *Logging*

`iparam.infeas_generic_names`

Controls whether generic names are used when an infeasible subproblem is created.

Default
off

Accepted
on, *off* (see *onoffkey*)

Example
```
task.putintparam(iparam.infeas_generic_names, onoffkey.off)
```

Generic name
MSK_IPAR_INFEAS_GENERIC_NAMES

Groups
Infeasibility report

`iparam.infeas_report_auto`

Controls whether an infeasibility report is automatically produced after the optimization if the problem is primal or dual infeasible.

Default
off

Accepted
on, *off* (see *onoffkey*)

Example
```
task.putintparam(iparam.infeas_report_auto, onoffkey.off)
```

Generic name
MSK_IPAR_INFEAS_REPORT_AUTO

Groups
Data input/output, *Solution input/output*

`iparam.infeas_report_level`

Controls the amount of information presented in an infeasibility report. Higher values imply more information.

Default
1

Accepted
[0; +inf]

Example
```
task.putintparam(iparam.infeas_report_level, 1)
```

Generic name
MSK_IPAR_INFEAS_REPORT_LEVEL

Groups
Infeasibility report, *Output information*

`iparam.intpnt_basis`

Controls whether the interior-point optimizer also computes an optimal basis.

Default
always

Accepted
never, *always*, *no_error*, *if_feasible*, *reservered* (see *basindtype*)

Example
```
task.putintparam(iparam.intpnt_basis, basindtype.always)
```

See also
iparam.bi_ignore_max_iter, *iparam.bi_ignore_num_error*, *iparam.bi_max_iterations*, *iparam.bi_clean_optimizer*

Generic name
MSK_IPAR_INTPNT_BASIS

Groups
Interior-point method, *Basis identification*

`iparam.intpnt_diff_step`

Controls whether different step sizes are allowed in the primal and dual space.

Default
on

Accepted
- *on*: Different step sizes are allowed.
- *off*: Different step sizes are not allowed.

Example
```
task.putintparam(iparam.intpnt_diff_step, onoffkey.on)
```

Generic name
MSK_IPAR_INTPNT_DIFF_STEP

Groups
Interior-point method

`iparam.intpnt_hotstart`

Currently not in use.

Default
none

Accepted
none, *primal*, *dual*, *primal_dual* (see *intpnthotstart*)

Example
```
task.putintparam(iparam.intpnt_hotstart, intpnthotstart.none)
```

Generic name
MSK_IPAR_INTPNT_HOTSTART

Groups
Interior-point method

`iparam.intpnt_max_iterations`

Controls the maximum number of iterations allowed in the interior-point optimizer.

Default
400

Accepted
[0; +inf]

Example
```
task.putintparam(iparam.intpnt_max_iterations, 400)
```

Generic name
MSK_IPAR_INTPNT_MAX_ITERATIONS

Groups
Interior-point method, *Termination criteria*

`iparam.intpnt_max_num_cor`

Controls the maximum number of correctors allowed by the multiple corrector procedure. A negative value means that **MOSEK** is making the choice.

Default
-1

Accepted
[-1; +inf]

Example
```
task.putintparam(iparam.intpnt_max_num_cor, -1)
```

Generic name
MSK_IPAR_INTPNT_MAX_NUM_COR

Groups
Interior-point method

`iparam.intpnt_off_col_trh`

Controls how many offending columns are detected in the Jacobian of the constraint matrix.

0	no detection
1	aggressive detection
> 1	higher values mean less aggressive detection

Default
40

Accepted
[0; +inf]

Example
```
task.putintparam(iparam.intpnt_off_col_trh, 40)
```

Generic name
MSK_IPAR_INTPNT_OFF_COL_TRH

Groups
Interior-point method

`iparam.intpnt_order_gp_num_seeds`

The GP ordering is dependent on a random seed. Therefore, trying several random seeds may lead to a better ordering. This parameter controls the number of random seeds tried.

A value of 0 means that MOSEK makes the choice.

Default
0

Accepted
[0; +inf]

Example
```
task.putintparam(iparam.intpnt_order_gp_num_seeds, 0)
```

Generic name
MSK_IPAR_INTPNT_ORDER_GP_NUM_SEEDS

Groups
Interior-point method

`iparam.intpnt_order_method`

Controls the ordering strategy used by the interior-point optimizer when factorizing the Newton equation system.

Default
free

Accepted
free, *appminloc*, *experimental*, *try_graphpar*, *force_graphpar*, *none* (see *orderingtype*)

Example
```
task.putintparam(iparam.intpnt_order_method, orderingtype.free)
```

Generic name
MSK_IPAR_INTPNT_ORDER_METHOD

Groups
Interior-point method

`iparam.intpnt_regularization_use`

Controls whether regularization is allowed.

Default
on

Accepted
on, *off* (see *onoffkey*)

Example
`task.putintparam(iparam.intpnt_regularization_use, onoffkey.on)`
Generic name
MSK_IPAR_INTPNT_REGULARIZATION_USE
Groups
Interior-point method

`iparam.intpnt_scaling`

Controls how the problem is scaled before the interior-point optimizer is used.

Default
free
Accepted
free, *none* (see *scalingtype*)
Example
`task.putintparam(iparam.intpnt_scaling, scalingtype.free)`
Generic name
MSK_IPAR_INTPNT_SCALING
Groups
Interior-point method

`iparam.intpnt_solve_form`

Controls whether the primal or the dual problem is solved.

Default
free
Accepted
free, *primal*, *dual* (see *solveform*)
Example
`task.putintparam(iparam.intpnt_solve_form, solveform.free)`
Generic name
MSK_IPAR_INTPNT_SOLVE_FORM
Groups
Interior-point method

`iparam.intpnt_starting_point`

Starting point used by the interior-point optimizer.

Default
free
Accepted
free, *guess*, *constant* (see *startpointtype*)
Example
`task.putintparam(iparam.intpnt_starting_point, startpointtype.free)`
Generic name
MSK_IPAR_INTPNT_STARTING_POINT
Groups
Interior-point method

`iparam.license_debug`

This option is used to turn on debugging of the license manager.

Default
off
Accepted
on, *off* (see *onoffkey*)
Example
`task.putintparam(iparam.license_debug, onoffkey.off)`

Generic name
MSK_IPAR_LICENSE_DEBUG

Groups
License manager

`iparam.license_pause_time`

If *iparam.license_wait* is *onoffkey.on* and no license is available, then **MOSEK** sleeps a number of milliseconds between each check of whether a license has become free.

Default
100

Accepted
[0; 1000000]

Example
```
task.putintparam(iparam.license_pause_time, 100)
```

Generic name
MSK_IPAR_LICENSE_PAUSE_TIME

Groups
License manager

`iparam.license_suppress_expire_wrns`

Controls whether license features expire warnings are suppressed.

Default
off

Accepted
on, *off* (see *onoffkey*)

Example
```
task.putintparam(iparam.license_suppress_expire_wrns, onoffkey.off)
```

Generic name
MSK_IPAR_LICENSE_SUPPRESS_EXPIRE_WRNS

Groups
License manager, *Output information*

`iparam.license_trh_expiry_wrn`

If a license feature expires in a numbers of days less than the value of this parameter then a warning will be issued.

Default
7

Accepted
[0; +inf]

Example
```
task.putintparam(iparam.license_trh_expiry_wrn, 7)
```

Generic name
MSK_IPAR_LICENSE_TRH_EXPIRY_WRN

Groups
License manager, *Output information*

`iparam.license_wait`

If all licenses are in use **MOSEK** returns with an error code. However, by turning on this parameter **MOSEK** will wait for an available license.

Default
off

Accepted
on, *off* (see *onoffkey*)

Example
`task.putintparam(iparam.license_wait, onoffkey.off)`

Generic name
MSK_IPAR_LICENSE_WAIT

Groups
Overall solver, *Overall system*, *License manager*

`iparam.log`

Controls the amount of log information. The value 0 implies that all log information is suppressed. A higher level implies that more information is logged.

Please note that if a task is employed to solve a sequence of optimization problems the value of this parameter is reduced by the value of *iparam.log_cut_second_opt* for the second and any subsequent optimizations.

Default
10

Accepted
[0; +inf]

Example
`task.putintparam(iparam.log, 10)`

See also
iparam.log_cut_second_opt

Generic name
MSK_IPAR_LOG

Groups
Output information, *Logging*

`iparam.log_ana_pro`

Controls amount of output from the problem analyzer.

Default
1

Accepted
[0; +inf]

Example
`task.putintparam(iparam.log_ana_pro, 1)`

Generic name
MSK_IPAR_LOG_ANA_PRO

Groups
Analysis, *Logging*

`iparam.log_bi`

Controls the amount of output printed by the basis identification procedure. A higher level implies that more information is logged.

Default
1

Accepted
[0; +inf]

Example
`task.putintparam(iparam.log_bi, 1)`

Generic name
MSK_IPAR_LOG_BI

Groups
Basis identification, *Output information*, *Logging*

`iparam.log_bi_freq`

Controls how frequently the optimizer outputs information about the basis identification and how frequent the user-defined callback function is called.

Default
2500

Accepted
[0; +inf]

Example
```
task.putintparam(iparam.log_bi_freq, 2500)
```

Generic name
MSK_IPAR_LOG_BI_FREQ

Groups
Basis identification, *Output information*, *Logging*

`iparam.log_cut_second_opt`

If a task is employed to solve a sequence of optimization problems, then the value of the log levels is reduced by the value of this parameter. E.g *iparam.log* and *iparam.log_sim* are reduced by the value of this parameter for the second and any subsequent optimizations.

Default
1

Accepted
[0; +inf]

Example
```
task.putintparam(iparam.log_cut_second_opt, 1)
```

See also
iparam.log, *iparam.log_intpnt*, *iparam.log_mio*, *iparam.log_sim*

Generic name
MSK_IPAR_LOG_CUT_SECOND_OPT

Groups
Output information, *Logging*

`iparam.log_expand`

Controls the amount of logging when a data item such as the maximum number constrains is expanded.

Default
1

Accepted
[0; +inf]

Example
```
task.putintparam(iparam.log_expand, 1)
```

Generic name
MSK_IPAR_LOG_EXPAND

Groups
Output information, *Logging*

`iparam.log_feas_repair`

Controls the amount of output printed when performing feasibility repair. A value higher than one means extensive logging.

Default
1

Accepted
[0; +inf]

Example
```
task.putintparam(iparam.log_feas_repair, 1)
```

Generic name
MSK_IPAR_LOG_FEAS_REPAIR

Groups
Output information, *Logging*

`iparam.log_file`

If turned on, then some log info is printed when a file is written or read.

Default
1

Accepted
[0; +inf]

Example
```
task.putintparam(iparam.log_file, 1)
```

Generic name
MSK_IPAR_LOG_FILE

Groups
Data input/output, *Output information*, *Logging*

`iparam.log_include_summary`

If on, then the solution summary will be printed by *Task.optimize*, so a separate call to *Task.solutionsummary* is not necessary.

Default
off

Accepted
on, *off* (see *onoffkey*)

Example
```
task.putintparam(iparam.log_include_summary, onoffkey.off)
```

Generic name
MSK_IPAR_LOG_INCLUDE_SUMMARY

Groups
Output information, *Logging*

`iparam.log_infeas_ana`

Controls amount of output printed by the infeasibility analyzer procedures. A higher level implies that more information is logged.

Default
1

Accepted
[0; +inf]

Example
```
task.putintparam(iparam.log_infeas_ana, 1)
```

Generic name
MSK_IPAR_LOG_INFEAS_ANA

Groups
Infeasibility report, *Output information*, *Logging*

`iparam.log_intpnt`

Controls amount of output printed by the interior-point optimizer. A higher level implies that more information is logged.

Default
1

Accepted
[0; +inf]

Example
`task.putintparam(iparam.log_intpnt, 1)`
Generic name
MSK_IPAR_LOG_INTPNT
Groups
Interior-point method, *Output information*, *Logging*

`iparam.log_local_info`

Controls whether local identifying information like environment variables, filenames, IP addresses etc. are printed to the log.

Note that this will only affect some functions. Some functions that specifically emit system information will not be affected.

Default
on
Accepted
on, *off* (see *onoffkey*)
Example
`task.putintparam(iparam.log_local_info, onoffkey.on)`
Generic name
MSK_IPAR_LOG_LOCAL_INFO
Groups
Output information, *Logging*

`iparam.log_mio`

Controls the log level for the mixed-integer optimizer. A higher level implies that more information is logged.

Default
4
Accepted
[0; +inf]
Example
`task.putintparam(iparam.log_mio, 4)`
Generic name
MSK_IPAR_LOG_MIO
Groups
Mixed-integer optimization, *Output information*, *Logging*

`iparam.log_mio_freq`

Controls how frequent the mixed-integer optimizer prints the log line. It will print line every time *iparam.log_mio_freq* relaxations have been solved.

Default
10
Accepted
[-inf; +inf]
Example
`task.putintparam(iparam.log_mio_freq, 10)`
Generic name
MSK_IPAR_LOG_MIO_FREQ
Groups
Mixed-integer optimization, *Output information*, *Logging*

`iparam.log_order`

If turned on, then factor lines are added to the log.

Default
1
Accepted
[0; +inf]
Example
`task.putintparam(iparam.log_order, 1)`
Generic name
MSK_IPAR_LOG_ORDER
Groups
Output information, *Logging*

`iparam.log_presolve`

Controls amount of output printed by the presolve procedure. A higher level implies that more information is logged.

Default
1
Accepted
[0; +inf]
Example
`task.putintparam(iparam.log_presolve, 1)`
Generic name
MSK_IPAR_LOG_PRESOLVE
Groups
Logging

`iparam.log_sensitivity`

Controls the amount of logging during the sensitivity analysis.

- 0. Means no logging information is produced.
- 1. Timing information is printed.
- 2. Sensitivity results are printed.

Default
1
Accepted
[0; +inf]
Example
`task.putintparam(iparam.log_sensitivity, 1)`
Generic name
MSK_IPAR_LOG_SENSITIVITY
Groups
Output information, *Logging*

`iparam.log_sensitivity_opt`

Controls the amount of logging from the optimizers employed during the sensitivity analysis. 0 means no logging information is produced.

Default
0
Accepted
[0; +inf]
Example
`task.putintparam(iparam.log_sensitivity_opt, 0)`
Generic name
MSK_IPAR_LOG_SENSITIVITY_OPT
Groups
Output information, *Logging*

`iparam.log_sim`

Controls amount of output printed by the simplex optimizer. A higher level implies that more information is logged.

Default
4

Accepted
[0; +inf]

Example
```
task.putintparam(iparam.log_sim, 4)
```

Generic name
MSK_IPAR_LOG_SIM

Groups
Simplex optimizer, *Output information*, *Logging*

`iparam.log_sim_freq`

Controls how frequent the simplex optimizer outputs information about the optimization and how frequent the user-defined callback function is called.

Default
1000

Accepted
[0; +inf]

Example
```
task.putintparam(iparam.log_sim_freq, 1000)
```

Generic name
MSK_IPAR_LOG_SIM_FREQ

Groups
Simplex optimizer, *Output information*, *Logging*

`iparam.log_sim_freq_giga_ticks`

Controls how frequent the new simplex optimizer outputs information about the optimization and how frequent the user-defined callback function is called.

- -1. Logging is disabled.
- 0. Logging at highest frequency (every iteration).
- ≥ 1. Logging at given frequency measured in giga ticks.

Default
100

Accepted
[-1; +inf]

Example
```
task.putintparam(iparam.log_sim_freq_giga_ticks, 100)
```

Generic name
MSK_IPAR_LOG_SIM_FREQ_GIGA_TICKS

Groups
Simplex optimizer, *Output information*, *Logging*

`iparam.log_storage`

When turned on, **MOSEK** prints messages regarding the storage usage and allocation.

Default
0

Accepted
[0; +inf]

Example
```
task.putintparam(iparam.log_storage, 0)
```

Generic name
MSK_IPAR_LOG_STORAGE

Groups
Output information, *Overall system*, *Logging*

`iparam.max_num_warnings`

Each warning is shown a limited number of times controlled by this parameter. A negative value is identical to infinite number of times.

Default
10

Accepted
[-inf; +inf]

Example
```
task.putintparam(iparam.max_num_warnings, 10)
```

Generic name
MSK_IPAR_MAX_NUM_WARNINGS

Groups
Output information

`iparam.mio_branch_dir`

Controls whether the mixed-integer optimizer is branching up or down by default.

Default
free

Accepted
free, *up*, *down*, *near*, *far*, *root_lp*, *guided*, *pseudocost* (see *branchdir*)

Example
```
task.putintparam(iparam.mio_branch_dir, branchdir.free)
```

Generic name
MSK_IPAR_MIO_BRANCH_DIR

Groups
Mixed-integer optimization

`iparam.mio_conflict_analysis_level`

Controls the amount of conflict analysis employed by the mixed-integer optimizer.

- -1. The optimizer chooses the level of conflict analysis employed
- 0. conflict analysis is disabled
- 1. A lower amount of conflict analysis is employed
- 2. A higher amount of conflict analysis is employed

Default
-1

Accepted
[-1; 2]

Example
```
task.putintparam(iparam.mio_conflict_analysis_level, -1)
```

Generic name
MSK_IPAR_MIO_CONFLICT_ANALYSIS_LEVEL

Groups
Mixed-integer optimization

`iparam.mio_conic_outer_approximation`

If this option is turned on outer approximation is used when solving relaxations of conic problems; otherwise interior point is used.

Default
off
Accepted
on, *off* (see *onoffkey*)
Example
`task.putintparam(iparam.mio_conic_outer_approximation, onoffkey.off)`
Generic name
MSK_IPAR_MIO_CONIC_OUTER_APPROXIMATION
Groups
Mixed-integer optimization

`iparam.mio_construct_sol`

If set to *onoffkey.on* and all integer variables have been given a value for which a feasible mixed integer solution exists, then **MOSEK** generates an initial solution to the mixed integer problem by fixing all integer values and solving the remaining problem.

Default
off
Accepted
on, *off* (see *onoffkey*)
Example
`task.putintparam(iparam.mio_construct_sol, onoffkey.off)`
Generic name
MSK_IPAR_MIO_CONSTRUCT_SOL
Groups
Mixed-integer optimization

`iparam.mio_crossover_max_nodes`

Controls the maximum number of nodes allowed in each call to the Crossover heuristic. The default value of -1 means that the value is determined automatically. A value of zero turns off the heuristic.

Default
-1
Accepted
[-1; +inf]
Example
`task.putintparam(iparam.mio_crossover_max_nodes, -1)`
Generic name
MSK_IPAR_MIO_CROSSOVER_MAX_NODES
Groups
Mixed-integer optimization

`iparam.mio_cut_clique`

Controls whether clique cuts should be generated.

Default
on
Accepted
on, *off* (see *onoffkey*)
Example
`task.putintparam(iparam.mio_cut_clique, onoffkey.on)`
Generic name
MSK_IPAR_MIO_CUT_CLIQUE
Groups
Mixed-integer optimization

`iparam.mio_cut_cmir`

Controls whether mixed integer rounding cuts should be generated.

Default
on

Accepted
on, *off* (see *onoffkey*)

Example
```
task.putintparam(iparam.mio_cut_cmir, onoffkey.on)
```

Generic name
MSK_IPAR_MIO_CUT_CMIR

Groups
Mixed-integer optimization

`iparam.mio_cut_gmi`

Controls whether GMI cuts should be generated.

Default
on

Accepted
on, *off* (see *onoffkey*)

Example
```
task.putintparam(iparam.mio_cut_gmi, onoffkey.on)
```

Generic name
MSK_IPAR_MIO_CUT_GMI

Groups
Mixed-integer optimization

`iparam.mio_cut_implied_bound`

Controls whether implied bound cuts should be generated.

Default
on

Accepted
on, *off* (see *onoffkey*)

Example
```
task.putintparam(iparam.mio_cut_implied_bound, onoffkey.on)
```

Generic name
MSK_IPAR_MIO_CUT_IMPLIED_BOUND

Groups
Mixed-integer optimization

`iparam.mio_cut_knapsack_cover`

Controls whether knapsack cover cuts should be generated.

Default
on

Accepted
on, *off* (see *onoffkey*)

Example
```
task.putintparam(iparam.mio_cut_knapsack_cover, onoffkey.on)
```

Generic name
MSK_IPAR_MIO_CUT_KNAPSACK_COVER

Groups
Mixed-integer optimization

`iparam.mio_cut_lipro`

Controls whether lift-and-project cuts should be generated.

Default
off

Accepted
on, *off* (see *onoffkey*)

Example
```
task.putintparam(iparam.mio_cut_lipro, onoffkey.off)
```

Generic name
MSK_IPAR_MIO_CUT_LIPRO

Groups
Mixed-integer optimization

`iparam.mio_cut_selection_level`

Controls how aggressively generated cuts are selected to be included in the relaxation.

- −1. The optimizer chooses the level of cut selection
- 0. Generated cuts less likely to be added to the relaxation
- 1. Cuts are more aggressively selected to be included in the relaxation

Default
-1

Accepted
[-1; +1]

Example
```
task.putintparam(iparam.mio_cut_selection_level, -1)
```

Generic name
MSK_IPAR_MIO_CUT_SELECTION_LEVEL

Groups
Mixed-integer optimization

`iparam.mio_data_permutation_method`

Controls what problem data permutation method is appplied to mixed-integer problems.

Default
none

Accepted
none, *cyclic_shift*, *random* (see *miodatapermmethod*)

Example
```
task.putintparam(iparam.mio_data_permutation_method,
miodatapermmethod.none)
```

Generic name
MSK_IPAR_MIO_DATA_PERMUTATION_METHOD

Groups
Mixed-integer optimization

`iparam.mio_dual_ray_analysis_level`

Controls the amount of dual ray analysis employed by the mixed-integer optimizer.

- −1. The optimizer chooses the level of dual ray analysis employed
- 0. Dual ray analysis is disabled
- 1. A lower amount of dual ray analysis is employed
- 2. A higher amount of dual ray analysis is employed

Default
-1

Accepted
[-1; 2]

Example
```
task.putintparam(iparam.mio_dual_ray_analysis_level, -1)
```

Generic name
MSK_IPAR_MIO_DUAL_RAY_ANALYSIS_LEVEL

Groups
Mixed-integer optimization

`iparam.mio_feaspump_level`

Controls the way the Feasibility Pump heuristic is employed by the mixed-integer optimizer.

- −1. The optimizer chooses how the Feasibility Pump is used
- 0. The Feasibility Pump is disabled
- 1. The Feasibility Pump is enabled with an effort to improve solution quality
- 2. The Feasibility Pump is enabled with an effort to reach feasibility early

Default
-1

Accepted
[-1; 2]

Example
```
task.putintparam(iparam.mio_feaspump_level, -1)
```

Generic name
MSK_IPAR_MIO_FEASPUMP_LEVEL

Groups
Mixed-integer optimization

`iparam.mio_heuristic_level`

Controls the heuristic employed by the mixed-integer optimizer to locate an initial good integer feasible solution. A value of zero means the heuristic is not used at all. A larger value than 0 means that a gradually more sophisticated heuristic is used which is computationally more expensive. A negative value implies that the optimizer chooses the heuristic. Normally a value around 3 to 5 should be optimal.

Default
-1

Accepted
[-inf; +inf]

Example
```
task.putintparam(iparam.mio_heuristic_level, -1)
```

Generic name
MSK_IPAR_MIO_HEURISTIC_LEVEL

Groups
Mixed-integer optimization

`iparam.mio_independent_block_level`

Controls the way the mixed-integer optimizer tries to find and exploit a decomposition of the problem into independent blocks.

- −1. The optimizer chooses how independent-block structure is handled
- 0. No independent-block structure is detected
- 1. Independent-block structure may be exploited only in presolve
- 2. Independent-block structure may be exploited through a dedicated algorithm after the root node

- 3. Independent-block structure may be exploited through a dedicated algorithm before the root node

Default
-1
Accepted
[-1; 3]
Example
```
task.putintparam(iparam.mio_independent_block_level, -1)
```
Generic name
MSK_IPAR_MIO_INDEPENDENT_BLOCK_LEVEL
Groups
Mixed-integer optimization

`iparam.mio_max_num_branches`

Maximum number of branches allowed during the branch and bound search. A negative value means infinite.

Default
-1
Accepted
[-inf; +inf]
Example
```
task.putintparam(iparam.mio_max_num_branches, -1)
```
Generic name
MSK_IPAR_MIO_MAX_NUM_BRANCHES
Groups
Mixed-integer optimization, *Termination criteria*

`iparam.mio_max_num_relaxs`

Maximum number of relaxations allowed during the branch and bound search. A negative value means infinite.

Default
-1
Accepted
[-inf; +inf]
Example
```
task.putintparam(iparam.mio_max_num_relaxs, -1)
```
Generic name
MSK_IPAR_MIO_MAX_NUM_RELAXS
Groups
Mixed-integer optimization

`iparam.mio_max_num_restarts`

Maximum number of restarts allowed during the branch and bound search.

Default
10
Accepted
[0; +inf]
Example
```
task.putintparam(iparam.mio_max_num_restarts, 10)
```
Generic name
MSK_IPAR_MIO_MAX_NUM_RESTARTS
Groups
Mixed-integer optimization

`iparam.mio_max_num_root_cut_rounds`

Maximum number of cut separation rounds at the root node.

Default
100

Accepted
[0; +inf]

Example
```
task.putintparam(iparam.mio_max_num_root_cut_rounds, 100)
```

Generic name
MSK_IPAR_MIO_MAX_NUM_ROOT_CUT_ROUNDS

Groups
Mixed-integer optimization, *Termination criteria*

`iparam.mio_max_num_solutions`

The mixed-integer optimizer can be terminated after a certain number of different feasible solutions has been located. If this parameter has the value $n > 0$, then the mixed-integer optimizer will be terminated when n feasible solutions have been located.

Default
-1

Accepted
[-inf; +inf]

Example
```
task.putintparam(iparam.mio_max_num_solutions, -1)
```

Generic name
MSK_IPAR_MIO_MAX_NUM_SOLUTIONS

Groups
Mixed-integer optimization, *Termination criteria*

`iparam.mio_memory_emphasis_level`

Controls how much emphasis is put on reducing memory usage. Being more conservative about memory usage may come at the cost of decreased solution speed.

- 0. The optimizer chooses
- 1. More emphasis is put on reducing memory usage and less on speed

Default
0

Accepted
[0; +1]

Example
```
task.putintparam(iparam.mio_memory_emphasis_level, 0)
```

Generic name
MSK_IPAR_MIO_MEMORY_EMPHASIS_LEVEL

Groups
Mixed-integer optimization

`iparam.mio_min_rel`

Number of times a variable must have been branched on for its pseudocost to be considered reliable.

Default
5

Accepted
[0; +inf]

Example
```
task.putintparam(iparam.mio_min_rel, 5)
```

Generic name
MSK_IPAR_MIO_MIN_REL

Groups
Mixed-integer optimization

`iparam.mio_mode`

Controls whether the optimizer includes the integer restrictions and disjunctive constraints when solving a (mixed) integer optimization problem.

Default
satisfied

Accepted
ignored, *satisfied* (see *miomode*)

Example
```
task.putintparam(iparam.mio_mode, miomode.satisfied)
```

Generic name
MSK_IPAR_MIO_MODE

Groups
Overall solver

`iparam.mio_node_optimizer`

Controls which optimizer is employed at the non-root nodes in the mixed-integer optimizer.

Default
free

Accepted
free, *intpnt*, *conic*, *primal_simplex*, *dual_simplex*, *new_primal_simplex*, *new_dual_simplex*, *free_simplex*, *mixed_int* (see *optimizertype*)

Example
```
task.putintparam(iparam.mio_node_optimizer, optimizertype.free)
```

Generic name
MSK_IPAR_MIO_NODE_OPTIMIZER

Groups
Mixed-integer optimization

`iparam.mio_node_selection`

Controls the node selection strategy employed by the mixed-integer optimizer.

Default
free

Accepted
free, *first*, *best*, *pseudo* (see *mionodeseltype*)

Example
```
task.putintparam(iparam.mio_node_selection, mionodeseltype.free)
```

Generic name
MSK_IPAR_MIO_NODE_SELECTION

Groups
Mixed-integer optimization

`iparam.mio_numerical_emphasis_level`

Controls how much emphasis is put on reducing numerical problems possibly at the expense of solution speed.

- 0. The optimizer chooses
- 1. More emphasis is put on reducing numerical problems
- 2. Even more emphasis

Default
0

Accepted
[0; +2]

Example
```
task.putintparam(iparam.mio_numerical_emphasis_level, 0)
```

Generic name
MSK_IPAR_MIO_NUMERICAL_EMPHASIS_LEVEL

Groups
Mixed-integer optimization

`iparam.mio_opt_face_max_nodes`

Controls the maximum number of nodes allowed in each call to the optimal face heuristic. The default value of -1 means that the value is determined automatically. A value of zero turns off the heuristic.

Default
-1

Accepted
[-1; +inf]

Example
```
task.putintparam(iparam.mio_opt_face_max_nodes, -1)
```

Generic name
MSK_IPAR_MIO_OPT_FACE_MAX_NODES

Groups
Mixed-integer optimization

`iparam.mio_perspective_reformulate`

Enables or disables perspective reformulation in presolve.

Default
on

Accepted
on, *off* (see *onoffkey*)

Example
```
task.putintparam(iparam.mio_perspective_reformulate, onoffkey.on)
```

Generic name
MSK_IPAR_MIO_PERSPECTIVE_REFORMULATE

Groups
Mixed-integer optimization

`iparam.mio_presolve_aggregator_use`

Controls if the aggregator should be used.

Default
on

Accepted
on, *off* (see *onoffkey*)

Example
```
task.putintparam(iparam.mio_presolve_aggregator_use, onoffkey.on)
```

Generic name
MSK_IPAR_MIO_PRESOLVE_AGGREGATOR_USE

Groups
Presolve

`iparam.mio_probing_level`

Controls the amount of probing employed by the mixed-integer optimizer in presolve.

- -1. The optimizer chooses the level of probing employed
- 0. Probing is disabled
- 1. A low amount of probing is employed
- 2. A medium amount of probing is employed
- 3. A high amount of probing is employed

Default
-1
Accepted
[-1; 3]
Example
`task.putintparam(iparam.mio_probing_level, -1)`
Generic name
MSK_IPAR_MIO_PROBING_LEVEL
Groups
Mixed-integer optimization

`iparam.mio_propagate_objective_constraint`

Use objective domain propagation.

Default
off
Accepted
on, *off* (see *onoffkey*)
Example
`task.putintparam(iparam.mio_propagate_objective_constraint, onoffkey.off)`
Generic name
MSK_IPAR_MIO_PROPAGATE_OBJECTIVE_CONSTRAINT
Groups
Mixed-integer optimization

`iparam.mio_qcqo_reformulation_method`

Controls what reformulation method is applied to mixed-integer quadratic problems.

Default
free
Accepted
free, *none*, *linearization*, *eigen_val_method*, *diag_sdp*, *relax_sdp* (see *miqcqoreformmethod*)
Example
`task.putintparam(iparam.mio_qcqo_reformulation_method, miqcqoreformmethod.free)`
Generic name
MSK_IPAR_MIO_QCQO_REFORMULATION_METHOD
Groups
Mixed-integer optimization

`iparam.mio_rens_max_nodes`

Controls the maximum number of nodes allowed in each call to the RENS heuristic. The default value of -1 means that the value is determined automatically. A value of zero turns off the heuristic.

Default
-1

Accepted
[-1; +inf]
Example
`task.putintparam(iparam.mio_rens_max_nodes, -1)`
Generic name
MSK_IPAR_MIO_RENS_MAX_NODES
Groups
Mixed-integer optimization

`iparam.mio_rins_max_nodes`

Controls the maximum number of nodes allowed in each call to the RINS heuristic. The default value of -1 means that the value is determined automatically. A value of zero turns off the heuristic.

Default
-1
Accepted
[-1; +inf]
Example
`task.putintparam(iparam.mio_rins_max_nodes, -1)`
Generic name
MSK_IPAR_MIO_RINS_MAX_NODES
Groups
Mixed-integer optimization

`iparam.mio_root_optimizer`

Controls which optimizer is employed at the root node in the mixed-integer optimizer.

Default
free
Accepted
free, *intpnt*, *conic*, *primal_simplex*, *dual_simplex*, *new_primal_simplex*, *new_dual_simplex*, *free_simplex*, *mixed_int* (see *optimizertype*)
Example
`task.putintparam(iparam.mio_root_optimizer, optimizertype.free)`
Generic name
MSK_IPAR_MIO_ROOT_OPTIMIZER
Groups
Mixed-integer optimization

`iparam.mio_seed`

Sets the random seed used for randomization in the mixed integer optimizer. Selecting a different seed can change the path the optimizer takes to the optimal solution.

Default
42
Accepted
[0; +inf]
Example
`task.putintparam(iparam.mio_seed, 42)`
Generic name
MSK_IPAR_MIO_SEED
Groups
Mixed-integer optimization

`iparam.mio_symmetry_level`

Controls the amount of symmetry detection and handling employed by the mixed-integer optimizer in presolve.

- −1. The optimizer chooses the level of symmetry detection and handling employed
- 0. Symmetry detection and handling is disabled
- 1. A low amount of symmetry detection and handling is employed
- 2. A medium amount of symmetry detection and handling is employed
- 3. A high amount of symmetry detection and handling is employed
- 4. An extremely high amount of symmetry detection and handling is employed

Default
-1
Accepted
[-1; 4]
Example
```
task.putintparam(iparam.mio_symmetry_level, -1)
```
Generic name
MSK_IPAR_MIO_SYMMETRY_LEVEL
Groups
Mixed-integer optimization

`iparam.mio_var_selection`

Controls the variable selection strategy employed by the mixed-integer optimizer.

Default
free
Accepted
free, *pseudocost*, *strong* (see *miovarseltype*)
Example
```
task.putintparam(iparam.mio_var_selection, miovarseltype.free)
```
Generic name
MSK_IPAR_MIO_VAR_SELECTION
Groups
Mixed-integer optimization

`iparam.mio_vb_detection_level`

Controls how much effort is put into detecting variable bounds.

- −1. The optimizer chooses
- 0. No variable bounds are detected
- 1. Only detect variable bounds that are directly represented in the problem
- 2. Detect variable bounds in probing

Default
-1
Accepted
[-1; +2]
Example
```
task.putintparam(iparam.mio_vb_detection_level, -1)
```
Generic name
MSK_IPAR_MIO_VB_DETECTION_LEVEL
Groups
Mixed-integer optimization

`iparam.mt_spincount`

Set the number of iterations to spin before sleeping.

Default
0

Accepted
[0; 1000000000]
Example
`task.putintparam(iparam.mt_spincount, 0)`
Generic name
MSK_IPAR_MT_SPINCOUNT
Groups
Overall system

`iparam.ng`

Not in use.

Default
off
Accepted
on, *off* (see *onoffkey*)
Example
`task.putintparam(iparam.ng, onoffkey.off)`
Generic name
MSK_IPAR_NG

`iparam.num_threads`

Controls the number of threads employed by the optimizer. If set to 0 then the number of threads is chosen by the optimizer, typically as minimum(number of cores, 32). If set to a positive value then exactly the selected number of threads will be used.

Default
0
Accepted
[0; +inf]
Example
`task.putintparam(iparam.num_threads, 0)`
Generic name
MSK_IPAR_NUM_THREADS
Groups
Overall system

`iparam.opf_write_header`

Write a text header with date and **MOSEK** version in an OPF file.

Default
on
Accepted
on, *off* (see *onoffkey*)
Example
`task.putintparam(iparam.opf_write_header, onoffkey.on)`
Generic name
MSK_IPAR_OPF_WRITE_HEADER
Groups
Data input/output

`iparam.opf_write_hints`

Write a hint section with problem dimensions in the beginning of an OPF file.

Default
on
Accepted
on, *off* (see *onoffkey*)

Example
`task.putintparam(iparam.opf_write_hints, onoffkey.on)`
Generic name
MSK_IPAR_OPF_WRITE_HINTS
Groups
Data input/output

`iparam.opf_write_line_length`

Aim to keep lines in OPF files not much longer than this.

Default
80
Accepted
[0; +inf]
Example
`task.putintparam(iparam.opf_write_line_length, 80)`
Generic name
MSK_IPAR_OPF_WRITE_LINE_LENGTH
Groups
Data input/output

`iparam.opf_write_parameters`

Write a parameter section in an OPF file.

Default
off
Accepted
on, *off* (see *onoffkey*)
Example
`task.putintparam(iparam.opf_write_parameters, onoffkey.off)`
Generic name
MSK_IPAR_OPF_WRITE_PARAMETERS
Groups
Data input/output

`iparam.opf_write_problem`

Write objective, constraints, bounds etc. to an OPF file.

Default
on
Accepted
on, *off* (see *onoffkey*)
Example
`task.putintparam(iparam.opf_write_problem, onoffkey.on)`
Generic name
MSK_IPAR_OPF_WRITE_PROBLEM
Groups
Data input/output

`iparam.opf_write_sol_bas`

If *iparam.opf_write_solutions* is *onoffkey.on* and a basic solution is defined, include the basic solution in OPF files.

Default
on
Accepted
on, *off* (see *onoffkey*)

Example
`task.putintparam(iparam.opf_write_sol_bas, onoffkey.on)`
Generic name
MSK_IPAR_OPF_WRITE_SOL_BAS
Groups
Data input/output

`iparam.opf_write_sol_itg`

If *iparam.opf_write_solutions* is *onoffkey.on* and an integer solution is defined, write the integer solution in OPF files.

Default
on
Accepted
on, *off* (see *onoffkey*)
Example
`task.putintparam(iparam.opf_write_sol_itg, onoffkey.on)`
Generic name
MSK_IPAR_OPF_WRITE_SOL_ITG
Groups
Data input/output

`iparam.opf_write_sol_itr`

If *iparam.opf_write_solutions* is *onoffkey.on* and an interior solution is defined, write the interior solution in OPF files.

Default
on
Accepted
on, *off* (see *onoffkey*)
Example
`task.putintparam(iparam.opf_write_sol_itr, onoffkey.on)`
Generic name
MSK_IPAR_OPF_WRITE_SOL_ITR
Groups
Data input/output

`iparam.opf_write_solutions`

Enable inclusion of solutions in the OPF files.

Default
off
Accepted
on, *off* (see *onoffkey*)
Example
`task.putintparam(iparam.opf_write_solutions, onoffkey.off)`
Generic name
MSK_IPAR_OPF_WRITE_SOLUTIONS
Groups
Data input/output

`iparam.optimizer`

The parameter controls which optimizer is used to optimize the task.

Default
free
Accepted
free, *intpnt*, *conic*, *primal_simplex*, *dual_simplex*, *new_primal_simplex*, *new_dual_simplex*, *free_simplex*, *mixed_int* (see *optimizertype*)

Example
`task.putintparam(iparam.optimizer, optimizertype.free)`
Generic name
MSK_IPAR_OPTIMIZER
Groups
Overall solver

`iparam.param_read_case_name`

If turned on, then names in the parameter file are case sensitive.

Default
on
Accepted
on, *off* (see *onoffkey*)
Example
`task.putintparam(iparam.param_read_case_name, onoffkey.on)`
Generic name
MSK_IPAR_PARAM_READ_CASE_NAME
Groups
Data input/output

`iparam.param_read_ign_error`

If turned on, then errors in parameter settings is ignored.

Default
off
Accepted
on, *off* (see *onoffkey*)
Example
`task.putintparam(iparam.param_read_ign_error, onoffkey.off)`
Generic name
MSK_IPAR_PARAM_READ_IGN_ERROR
Groups
Data input/output

`iparam.presolve_eliminator_max_fill`

Controls the maximum amount of fill-in that can be created by one pivot in the elimination phase of the presolve. A negative value means the parameter value is selected automatically.

Default
-1
Accepted
[-inf; +inf]
Example
`task.putintparam(iparam.presolve_eliminator_max_fill, -1)`
Generic name
MSK_IPAR_PRESOLVE_ELIMINATOR_MAX_FILL
Groups
Presolve

`iparam.presolve_eliminator_max_num_tries`

Control the maximum number of times the eliminator is tried. A negative value implies **MOSEK** decides.

Default
-1
Accepted
[-inf; +inf]

Example
`task.putintparam(iparam.presolve_eliminator_max_num_tries, -1)`
Generic name
MSK_IPAR_PRESOLVE_ELIMINATOR_MAX_NUM_TRIES
Groups
Presolve

`iparam.presolve_lindep_abs_work_trh`

Controls linear dependency check in presolve. The linear dependency check is potentially computationally expensive.

Default
100
Accepted
[-inf; +inf]
Example
`task.putintparam(iparam.presolve_lindep_abs_work_trh, 100)`
Generic name
MSK_IPAR_PRESOLVE_LINDEP_ABS_WORK_TRH
Groups
Presolve

`iparam.presolve_lindep_new`

Controls whether a new experimental linear dependency checker is employed.

Default
off
Accepted
on, *off* (see *onoffkey*)
Example
`task.putintparam(iparam.presolve_lindep_new, onoffkey.off)`
Generic name
MSK_IPAR_PRESOLVE_LINDEP_NEW
Groups
Presolve

`iparam.presolve_lindep_rel_work_trh`

Controls linear dependency check in presolve. The linear dependency check is potentially computationally expensive.

Default
100
Accepted
[-inf; +inf]
Example
`task.putintparam(iparam.presolve_lindep_rel_work_trh, 100)`
Generic name
MSK_IPAR_PRESOLVE_LINDEP_REL_WORK_TRH
Groups
Presolve

`iparam.presolve_lindep_use`

Controls whether the linear constraints are checked for linear dependencies.

Default
on
Accepted
on, *off* (see *onoffkey*)

Example
`task.putintparam(iparam.presolve_lindep_use, onoffkey.on)`
Generic name
MSK_IPAR_PRESOLVE_LINDEP_USE
Groups
Presolve

`iparam.presolve_max_num_pass`

Control the maximum number of times presolve passes over the problem. A negative value implies **MOSEK** decides.

Default
-1
Accepted
[-inf; +inf]
Example
`task.putintparam(iparam.presolve_max_num_pass, -1)`
Generic name
MSK_IPAR_PRESOLVE_MAX_NUM_PASS
Groups
Presolve

`iparam.presolve_max_num_reductions`

Controls the maximum number of reductions performed by the presolve. The value of the parameter is normally only changed in connection with debugging. A negative value implies that an infinite number of reductions are allowed.

Default
-1
Accepted
[-inf; +inf]
Example
`task.putintparam(iparam.presolve_max_num_reductions, -1)`
Generic name
MSK_IPAR_PRESOLVE_MAX_NUM_REDUCTIONS
Groups
Overall solver, *Presolve*

`iparam.presolve_use`

Controls whether the presolve is applied to a problem before it is optimized.

Default
free
Accepted
off, *on*, *free* (see *presolvemode*)
Example
`task.putintparam(iparam.presolve_use, presolvemode.free)`
Generic name
MSK_IPAR_PRESOLVE_USE
Groups
Overall solver, *Presolve*

`iparam.primal_repair_optimizer`

Controls which optimizer that is used to find the optimal repair.

Default
free

Accepted
free, *intpnt*, *conic*, *primal_simplex*, *dual_simplex*, *new_primal_simplex*, *new_dual_simplex*, *free_simplex*, *mixed_int* (see *optimizertype*)

Example
```
task.putintparam(iparam.primal_repair_optimizer, optimizertype.free)
```

Generic name
MSK_IPAR_PRIMAL_REPAIR_OPTIMIZER

Groups
Overall solver

`iparam.ptf_write_parameters`

If *iparam.ptf_write_parameters* is *onoffkey.on*, the parameters section is written.

Default
off

Accepted
on, *off* (see *onoffkey*)

Example
```
task.putintparam(iparam.ptf_write_parameters, onoffkey.off)
```

Generic name
MSK_IPAR_PTF_WRITE_PARAMETERS

Groups
Data input/output

`iparam.ptf_write_single_psd_terms`

Controls whether PSD terms with a coefficient matrix of just one non-zero are written as a single term instead of as a matrix term.

Default
off

Accepted
on, *off* (see *onoffkey*)

Example
```
task.putintparam(iparam.ptf_write_single_psd_terms, onoffkey.off)
```

Generic name
MSK_IPAR_PTF_WRITE_SINGLE_PSD_TERMS

Groups
Data input/output

`iparam.ptf_write_solutions`

If *iparam.ptf_write_solutions* is *onoffkey.on*, the solution section is written if any solutions are available, otherwise solution section is not written even if solutions are available.

Default
off

Accepted
on, *off* (see *onoffkey*)

Example
```
task.putintparam(iparam.ptf_write_solutions, onoffkey.off)
```

Generic name
MSK_IPAR_PTF_WRITE_SOLUTIONS

Groups
Data input/output

`iparam.ptf_write_transform`

If *iparam.ptf_write_transform* is *onoffkey.on*, constraint blocks with identifiable conic slacks are transformed into conic constraints and the slacks are eliminated.

Default
on

Accepted
on, *off* (see *onoffkey*)

Example
```
task.putintparam(iparam.ptf_write_transform, onoffkey.on)
```

Generic name
MSK_IPAR_PTF_WRITE_TRANSFORM

Groups
Data input/output

`iparam.read_async`

Controls whether files are read using synchronous or asynchronous reader.

Default
off

Accepted
- *on*: Use asynchronous reader
- *off*: Use synchronous reader

Example
```
task.putintparam(iparam.read_async, onoffkey.off)
```

Generic name
MSK_IPAR_READ_ASYNC

Groups
Data input/output

`iparam.read_debug`

Turns on additional debugging information when reading files.

Default
off

Accepted
on, *off* (see *onoffkey*)

Example
```
task.putintparam(iparam.read_debug, onoffkey.off)
```

Generic name
MSK_IPAR_READ_DEBUG

Groups
Data input/output

`iparam.read_keep_free_con`

Controls whether the free constraints are included in the problem. Applies to MPS files.

Default
off

Accepted
- *on*: The free constraints are kept.
- *off*: The free constraints are discarded.

Example
```
task.putintparam(iparam.read_keep_free_con, onoffkey.off)
```

Generic name
MSK_IPAR_READ_KEEP_FREE_CON

Groups
Data input/output

`iparam.read_mps_format`

Controls how strictly the MPS file reader interprets the MPS format.

Default
free

Accepted
strict, *relaxed*, *free*, *cplex* (see *mpsformat*)

Example
```
task.putintparam(iparam.read_mps_format, mpsformat.free)
```

Generic name
MSK_IPAR_READ_MPS_FORMAT

Groups
Data input/output

`iparam.read_mps_width`

Controls the maximal number of characters allowed in one line of the MPS file.

Default
1024

Accepted
[80; +inf]

Example
```
task.putintparam(iparam.read_mps_width, 1024)
```

Generic name
MSK_IPAR_READ_MPS_WIDTH

Groups
Data input/output

`iparam.read_task_ignore_param`

Controls whether **MOSEK** should ignore the parameter setting defined in the task file and use the default parameter setting instead.

Default
off

Accepted
on, *off* (see *onoffkey*)

Example
```
task.putintparam(iparam.read_task_ignore_param, onoffkey.off)
```

Generic name
MSK_IPAR_READ_TASK_IGNORE_PARAM

Groups
Data input/output

`iparam.remote_use_compression`

Use compression when sending data to an optimization server.

Default
zstd

Accepted
none, *free*, *gzip*, *zstd* (see *compresstype*)

Example
```
task.putintparam(iparam.remote_use_compression, compresstype.zstd)
```

Generic name
MSK_IPAR_REMOTE_USE_COMPRESSION

`iparam.remove_unused_solutions`

Removes unused solutions before the optimization is performed.

Default
off

Accepted
on, *off* (see *onoffkey*)

Example
`task.putintparam(iparam.remove_unused_solutions, onoffkey.off)`

Generic name
MSK_IPAR_REMOVE_UNUSED_SOLUTIONS

Groups
Overall system

`iparam.sensitivity_all`

If set to *onoffkey.on*, then *Task.sensitivityreport* analyzes all bounds and variables instead of reading a specification from the file.

Default
off

Accepted
on, *off* (see *onoffkey*)

Example
`task.putintparam(iparam.sensitivity_all, onoffkey.off)`

Generic name
MSK_IPAR_SENSITIVITY_ALL

Groups
Overall solver

`iparam.sensitivity_type`

Controls which type of sensitivity analysis is to be performed.

Default
basis

Accepted
basis (see *sensitivitytype*)

Example
`task.putintparam(iparam.sensitivity_type, sensitivitytype.basis)`

Generic name
MSK_IPAR_SENSITIVITY_TYPE

Groups
Overall solver

`iparam.sim_basis_factor_use`

Controls whether an LU factorization of the basis is used in a hot-start. Forcing a refactorization sometimes improves the stability of the simplex optimizers, but in most cases there is a performance penalty.

Default
on

Accepted
on, *off* (see *onoffkey*)

Example
`task.putintparam(iparam.sim_basis_factor_use, onoffkey.on)`

Generic name
MSK_IPAR_SIM_BASIS_FACTOR_USE

Groups
Simplex optimizer

`iparam.sim_degen`

Controls how aggressively degeneration is handled.

Default
free

Accepted
none, *free*, *aggressive*, *moderate*, *minimum* (see *simdegen*)

Example
`task.putintparam(iparam.sim_degen, simdegen.free)`

Generic name
MSK_IPAR_SIM_DEGEN

Groups
Simplex optimizer

`iparam.sim_detect_pwl`

Not in use.

Default
on

Accepted
- *on*: PWL are detected.
- *off*: PWL are not detected.

Example
`task.putintparam(iparam.sim_detect_pwl, onoffkey.on)`

Generic name
MSK_IPAR_SIM_DETECT_PWL

Groups
Simplex optimizer

`iparam.sim_dual_crash`

Controls whether crashing is performed in the dual simplex optimizer. If this parameter is set to x, then a crash will be performed if a basis consists of more than $(100 - x)$ mod f_v entries, where f_v is the number of fixed variables.

Default
90

Accepted
[0; +inf]

Example
`task.putintparam(iparam.sim_dual_crash, 90)`

Generic name
MSK_IPAR_SIM_DUAL_CRASH

Groups
Dual simplex

`iparam.sim_dual_phaseone_method`

An experimental feature.

Default
0

Accepted
[0; 10]

Example
`task.putintparam(iparam.sim_dual_phaseone_method, 0)`

Generic name
MSK_IPAR_SIM_DUAL_PHASEONE_METHOD

Groups
Simplex optimizer

`iparam.sim_dual_restrict_selection`

The dual simplex optimizer can use a so-called restricted selection/pricing strategy to choose the outgoing variable. Hence, if restricted selection is applied, then the dual simplex optimizer first choose a subset of all the potential outgoing variables. Next, for some time it will choose the outgoing variable only among the subset. From time to time the subset is redefined. A larger value of this parameter implies that the optimizer will be more aggressive in its restriction strategy, i.e. a value of 0 implies that the restriction strategy is not applied at all.

Default
50

Accepted
[0; 100]

Example
```
task.putintparam(iparam.sim_dual_restrict_selection, 50)
```

Generic name
MSK_IPAR_SIM_DUAL_RESTRICT_SELECTION

Groups
Dual simplex

`iparam.sim_dual_selection`

Controls the choice of the incoming variable, known as the selection strategy, in the dual simplex optimizer.

Default
free

Accepted
free, *full*, *ase*, *devex*, *se*, *partial* (see *simseltype*)

Example
```
task.putintparam(iparam.sim_dual_selection, simseltype.free)
```

Generic name
MSK_IPAR_SIM_DUAL_SELECTION

Groups
Dual simplex

`iparam.sim_exploit_dupvec`

Controls if the simplex optimizers are allowed to exploit duplicated columns.

Default
off

Accepted
on, *off*, *free* (see *simdupvec*)

Example
```
task.putintparam(iparam.sim_exploit_dupvec, simdupvec.off)
```

Generic name
MSK_IPAR_SIM_EXPLOIT_DUPVEC

Groups
Simplex optimizer

`iparam.sim_hotstart`

Controls the type of hot-start that the simplex optimizer perform.

Default
free

Accepted
none, *free*, *status_keys* (see *simhotstart*)

Example
```
task.putintparam(iparam.sim_hotstart, simhotstart.free)
```

Generic name
MSK_IPAR_SIM_HOTSTART

Groups
Simplex optimizer

`iparam.sim_hotstart_lu`

Determines if the simplex optimizer should exploit the initial factorization.

Default
on

Accepted
- *on*: Factorization is reused if possible.
- *off*: Factorization is recomputed.

Example
```
task.putintparam(iparam.sim_hotstart_lu, onoffkey.on)
```

Generic name
MSK_IPAR_SIM_HOTSTART_LU

Groups
Simplex optimizer

`iparam.sim_max_iterations`

Maximum number of iterations that can be used by a simplex optimizer.

Default
10000000

Accepted
[0; +inf]

Example
```
task.putintparam(iparam.sim_max_iterations, 10000000)
```

Generic name
MSK_IPAR_SIM_MAX_ITERATIONS

Groups
Simplex optimizer, *Termination criteria*

`iparam.sim_max_num_setbacks`

Controls how many set-backs are allowed within a simplex optimizer. A set-back is an event where the optimizer moves in the wrong direction. This is impossible in theory but may happen due to numerical problems.

Default
250

Accepted
[0; +inf]

Example
```
task.putintparam(iparam.sim_max_num_setbacks, 250)
```

Generic name
MSK_IPAR_SIM_MAX_NUM_SETBACKS

Groups
Simplex optimizer

`iparam.sim_non_singular`

Controls if the simplex optimizer ensures a non-singular basis, if possible.

Default
on

Accepted
on, *off* (see *onoffkey*)

Example
`task.putintparam(iparam.sim_non_singular, onoffkey.on)`

Generic name
MSK_IPAR_SIM_NON_SINGULAR

Groups
Simplex optimizer

`iparam.sim_precision`

Experimental. Usage not recommended.

Default
normal

Accepted
normal, *extended* (see *simprecision*)

Example
`task.putintparam(iparam.sim_precision, simprecision.normal)`

Generic name
MSK_IPAR_SIM_PRECISION

Groups
Overall solver

`iparam.sim_precision_boost`

Controls whether the simplex optimizer is allowed to boost the precision during the computations if possible.

Default
off

Accepted
on, *off* (see *onoffkey*)

Example
`task.putintparam(iparam.sim_precision_boost, onoffkey.off)`

Generic name
MSK_IPAR_SIM_PRECISION_BOOST

Groups
Simplex optimizer

`iparam.sim_primal_crash`

Controls whether crashing is performed in the primal simplex optimizer. In general, if a basis consists of more than (100-this parameter value)% fixed variables, then a crash will be performed.

Default
90

Accepted
[0; +inf]

Example
`task.putintparam(iparam.sim_primal_crash, 90)`

Generic name
MSK_IPAR_SIM_PRIMAL_CRASH

Groups
Primal simplex

`iparam.sim_primal_phaseone_method`

An experimental feature.

Default
0

Accepted
[0; 10]

Example
`task.putintparam(iparam.sim_primal_phaseone_method, 0)`

Generic name
MSK_IPAR_SIM_PRIMAL_PHASEONE_METHOD

Groups
Simplex optimizer

`iparam.sim_primal_restrict_selection`

The primal simplex optimizer can use a so-called restricted selection/pricing strategy to choose the outgoing variable. Hence, if restricted selection is applied, then the primal simplex optimizer first choose a subset of all the potential incoming variables. Next, for some time it will choose the incoming variable only among the subset. From time to time the subset is redefined. A larger value of this parameter implies that the optimizer will be more aggressive in its restriction strategy, i.e. a value of 0 implies that the restriction strategy is not applied at all.

Default
50

Accepted
[0; 100]

Example
`task.putintparam(iparam.sim_primal_restrict_selection, 50)`

Generic name
MSK_IPAR_SIM_PRIMAL_RESTRICT_SELECTION

Groups
Primal simplex

`iparam.sim_primal_selection`

Controls the choice of the incoming variable, known as the selection strategy, in the primal simplex optimizer.

Default
free

Accepted
free, *full*, *ase*, *devex*, *se*, *partial* (see *simseltype*)

Example
`task.putintparam(iparam.sim_primal_selection, simseltype.free)`

Generic name
MSK_IPAR_SIM_PRIMAL_SELECTION

Groups
Primal simplex

`iparam.sim_refactor_freq`

Controls how frequent the basis is refactorized. The value 0 means that the optimizer determines the best point of refactorization. It is strongly recommended NOT to change this parameter.

Default
0

Accepted
[0; +inf]

Example
`task.putintparam(iparam.sim_refactor_freq, 0)`

Generic name
MSK_IPAR_SIM_REFACTOR_FREQ

Groups
Simplex optimizer

`iparam.sim_reformulation`

Controls if the simplex optimizers are allowed to reformulate the problem.

Default
off

Accepted
on, *off*, *free*, *aggressive* (see *simreform*)

Example
`task.putintparam(iparam.sim_reformulation, simreform.off)`

Generic name
MSK_IPAR_SIM_REFORMULATION

Groups
Simplex optimizer

`iparam.sim_save_lu`

Controls if the LU factorization stored should be replaced with the LU factorization corresponding to the initial basis.

Default
off

Accepted
on, *off* (see *onoffkey*)

Example
`task.putintparam(iparam.sim_save_lu, onoffkey.off)`

Generic name
MSK_IPAR_SIM_SAVE_LU

Groups
Simplex optimizer

`iparam.sim_scaling`

Controls how much effort is used in scaling the problem before a simplex optimizer is used.

Default
free

Accepted
free, *none* (see *scalingtype*)

Example
`task.putintparam(iparam.sim_scaling, scalingtype.free)`

Generic name
MSK_IPAR_SIM_SCALING

Groups
Simplex optimizer

`iparam.sim_scaling_method`

Controls how the problem is scaled before a simplex optimizer is used.

Default
pow2

Accepted
pow2, *free* (see *scalingmethod*)

Example
`task.putintparam(iparam.sim_scaling_method, scalingmethod.pow2)`

Generic name
MSK_IPAR_SIM_SCALING_METHOD

Groups
Simplex optimizer

`iparam.sim_seed`

Sets the random seed used for randomization in the simplex optimizers.

Default
23456
Accepted
[0; 32749]
Example
`task.putintparam(iparam.sim_seed, 23456)`
Generic name
MSK_IPAR_SIM_SEED
Groups
Simplex optimizer

`iparam.sim_solve_form`

Controls whether the primal or the dual problem is solved by the primal-/dual-simplex optimizer.

Default
free
Accepted
free, *primal*, *dual* (see *solveform*)
Example
`task.putintparam(iparam.sim_solve_form, solveform.free)`
Generic name
MSK_IPAR_SIM_SOLVE_FORM
Groups
Simplex optimizer

`iparam.sim_switch_optimizer`

The simplex optimizer sometimes chooses to solve the dual problem instead of the primal problem. This implies that if you have chosen to use the dual simplex optimizer and the problem is dualized, then it actually makes sense to use the primal simplex optimizer instead. If this parameter is on and the problem is dualized and furthermore the simplex optimizer is chosen to be the primal (dual) one, then it is switched to the dual (primal).

Default
off
Accepted
on, *off* (see *onoffkey*)
Example
`task.putintparam(iparam.sim_switch_optimizer, onoffkey.off)`
Generic name
MSK_IPAR_SIM_SWITCH_OPTIMIZER
Groups
Simplex optimizer

`iparam.sol_filter_keep_basic`

If turned on, then basic and super basic constraints and variables are written to the solution file independent of the filter setting.

Default
off
Accepted
on, *off* (see *onoffkey*)
Example
`task.putintparam(iparam.sol_filter_keep_basic, onoffkey.off)`
Generic name
MSK_IPAR_SOL_FILTER_KEEP_BASIC

Groups
Solution input/output

`iparam.sol_read_name_width`

When a solution is read by **MOSEK** and some constraint, variable or cone names contain blanks, then a maximum name width much be specified. A negative value implies that no name contain blanks.

Default
-1
Accepted
[-inf; +inf]
Example
```
task.putintparam(iparam.sol_read_name_width, -1)
```
Generic name
MSK_IPAR_SOL_READ_NAME_WIDTH
Groups
Data input/output, *Solution input/output*

`iparam.sol_read_width`

Controls the maximal acceptable width of line in the solutions when read by **MOSEK**.

Default
1024
Accepted
[80; +inf]
Example
```
task.putintparam(iparam.sol_read_width, 1024)
```
Generic name
MSK_IPAR_SOL_READ_WIDTH
Groups
Data input/output, *Solution input/output*

`iparam.timing_level`

Controls the amount of timing performed inside **MOSEK**.

Default
1
Accepted
[0; +inf]
Example
```
task.putintparam(iparam.timing_level, 1)
```
Generic name
MSK_IPAR_TIMING_LEVEL
Groups
Overall system

`iparam.write_async`

Controls whether files are read using synchronous or asynchronous writer.

Default
off
Accepted
- *on*: Use asynchronous writer
- *off*: Use synchronous writer

Example
```
task.putintparam(iparam.write_async, onoffkey.off)
```

Generic name
MSK_IPAR_WRITE_ASYNC
Groups
Data input/output

`iparam.write_bas_constraints`

Controls whether the constraint section is written to the basic solution file.

Default
on
Accepted
on, *off* (see *onoffkey*)
Example
`task.putintparam(iparam.write_bas_constraints, onoffkey.on)`
Generic name
MSK_IPAR_WRITE_BAS_CONSTRAINTS
Groups
Data input/output, *Solution input/output*

`iparam.write_bas_head`

Controls whether the header section is written to the basic solution file.

Default
on
Accepted
on, *off* (see *onoffkey*)
Example
`task.putintparam(iparam.write_bas_head, onoffkey.on)`
Generic name
MSK_IPAR_WRITE_BAS_HEAD
Groups
Data input/output, *Solution input/output*

`iparam.write_bas_variables`

Controls whether the variables section is written to the basic solution file.

Default
on
Accepted
on, *off* (see *onoffkey*)
Example
`task.putintparam(iparam.write_bas_variables, onoffkey.on)`
Generic name
MSK_IPAR_WRITE_BAS_VARIABLES
Groups
Data input/output, *Solution input/output*

`iparam.write_compression`

Controls whether the data file is compressed while it is written. 0 means no compression while higher values mean more compression.

Default
9
Accepted
[0; +inf]
Example
`task.putintparam(iparam.write_compression, 9)`

Generic name
MSK_IPAR_WRITE_COMPRESSION

Groups
Data input/output

`iparam.write_free_con`

Controls whether the free constraints are written to the data file. Applies to MPS files.

Default
on

Accepted
on, *off* (see *onoffkey*)

Example
```
task.putintparam(iparam.write_free_con, onoffkey.on)
```

Generic name
MSK_IPAR_WRITE_FREE_CON

Groups
Data input/output

`iparam.write_generic_names`

Controls whether generic names should be used instead of user-defined names when writing to the data file.

Default
off

Accepted
on, *off* (see *onoffkey*)

Example
```
task.putintparam(iparam.write_generic_names, onoffkey.off)
```

Generic name
MSK_IPAR_WRITE_GENERIC_NAMES

Groups
Data input/output

`iparam.write_ignore_incompatible_items`

Controls if the writer ignores incompatible problem items when writing files.

Default
off

Accepted
- *on*: Ignore items that cannot be written to the current output file format.
- *off*: Produce an error if the problem contains items that cannot the written to the current output file format.

Example
```
task.putintparam(iparam.write_ignore_incompatible_items, onoffkey.
off)
```

Generic name
MSK_IPAR_WRITE_IGNORE_INCOMPATIBLE_ITEMS

Groups
Data input/output

`iparam.write_int_constraints`

Controls whether the constraint section is written to the integer solution file.

Default
on

Accepted
on, *off* (see *onoffkey*)

Example
`task.putintparam(iparam.write_int_constraints, onoffkey.on)`
Generic name
MSK_IPAR_WRITE_INT_CONSTRAINTS
Groups
Data input/output, *Solution input/output*

`iparam.write_int_head`

Controls whether the header section is written to the integer solution file.

Default
on
Accepted
on, *off* (see *onoffkey*)
Example
`task.putintparam(iparam.write_int_head, onoffkey.on)`
Generic name
MSK_IPAR_WRITE_INT_HEAD
Groups
Data input/output, *Solution input/output*

`iparam.write_int_variables`

Controls whether the variables section is written to the integer solution file.

Default
on
Accepted
on, *off* (see *onoffkey*)
Example
`task.putintparam(iparam.write_int_variables, onoffkey.on)`
Generic name
MSK_IPAR_WRITE_INT_VARIABLES
Groups
Data input/output, *Solution input/output*

`iparam.write_json_indentation`

When set, the JSON task and solution files are written with indentation for better readability.

Default
off
Accepted
on, *off* (see *onoffkey*)
Example
`task.putintparam(iparam.write_json_indentation, onoffkey.off)`
Generic name
MSK_IPAR_WRITE_JSON_INDENTATION
Groups
Data input/output

`iparam.write_lp_full_obj`

Write all variables, including the ones with 0-coefficients, in the objective.

Default
on
Accepted
on, *off* (see *onoffkey*)
Example
`task.putintparam(iparam.write_lp_full_obj, onoffkey.on)`

Generic name
MSK_IPAR_WRITE_LP_FULL_OBJ

Groups
Data input/output

`iparam.write_lp_line_width`

Maximum width of line in an LP file written by **MOSEK**.

Default
80

Accepted
[40; +inf]

Example
```
task.putintparam(iparam.write_lp_line_width, 80)
```

Generic name
MSK_IPAR_WRITE_LP_LINE_WIDTH

Groups
Data input/output

`iparam.write_mps_format`

Controls in which format the MPS file is written.

Default
free

Accepted
strict, *relaxed*, *free*, *cplex* (see *mpsformat*)

Example
```
task.putintparam(iparam.write_mps_format, mpsformat.free)
```

Generic name
MSK_IPAR_WRITE_MPS_FORMAT

Groups
Data input/output

`iparam.write_mps_int`

Controls if marker records are written to the MPS file to indicate whether variables are integer restricted.

Default
on

Accepted
on, *off* (see *onoffkey*)

Example
```
task.putintparam(iparam.write_mps_int, onoffkey.on)
```

Generic name
MSK_IPAR_WRITE_MPS_INT

Groups
Data input/output

`iparam.write_sol_barvariables`

Controls whether the symmetric matrix variables section is written to the solution file.

Default
on

Accepted
on, *off* (see *onoffkey*)

Example
```
task.putintparam(iparam.write_sol_barvariables, onoffkey.on)
```

Generic name
MSK_IPAR_WRITE_SOL_BARVARIABLES
Groups
Data input/output, *Solution input/output*

`iparam.write_sol_constraints`
Controls whether the constraint section is written to the solution file.

Default
on
Accepted
on, *off* (see *onoffkey*)
Example
`task.putintparam(iparam.write_sol_constraints, onoffkey.on)`
Generic name
MSK_IPAR_WRITE_SOL_CONSTRAINTS
Groups
Data input/output, *Solution input/output*

`iparam.write_sol_head`
Controls whether the header section is written to the solution file.

Default
on
Accepted
on, *off* (see *onoffkey*)
Example
`task.putintparam(iparam.write_sol_head, onoffkey.on)`
Generic name
MSK_IPAR_WRITE_SOL_HEAD
Groups
Data input/output, *Solution input/output*

`iparam.write_sol_ignore_invalid_names`
Even if the names are invalid MPS names, then they are employed when writing the solution file.

Default
off
Accepted
on, *off* (see *onoffkey*)
Example
`task.putintparam(iparam.write_sol_ignore_invalid_names, onoffkey.off)`
Generic name
MSK_IPAR_WRITE_SOL_IGNORE_INVALID_NAMES
Groups
Data input/output, *Solution input/output*

`iparam.write_sol_variables`
Controls whether the variables section is written to the solution file.

Default
on
Accepted
on, *off* (see *onoffkey*)
Example
`task.putintparam(iparam.write_sol_variables, onoffkey.on)`
Generic name
MSK_IPAR_WRITE_SOL_VARIABLES
Groups
Data input/output, *Solution input/output*

15.7.3 String parameters

`sparam`

The enumeration type containing all string parameters.

`sparam.bas_sol_file_name`

Name of the `bas` solution file.

Accepted
Any valid file name.

Example
```
task.putstrparam(sparam.bas_sol_file_name, "somevalue")
```

Generic name
MSK_SPAR_BAS_SOL_FILE_NAME

Groups
Data input/output, *Solution input/output*

`sparam.data_file_name`

Data are read and written to this file.

Accepted
Any valid file name.

Example
```
task.putstrparam(sparam.data_file_name, "somevalue")
```

Generic name
MSK_SPAR_DATA_FILE_NAME

Groups
Data input/output

`sparam.debug_file_name`

MOSEK debug file.

Accepted
Any valid file name.

Example
```
task.putstrparam(sparam.debug_file_name, "somevalue")
```

Generic name
MSK_SPAR_DEBUG_FILE_NAME

Groups
Data input/output

`sparam.int_sol_file_name`

Name of the `int` solution file.

Accepted
Any valid file name.

Example
```
task.putstrparam(sparam.int_sol_file_name, "somevalue")
```

Generic name
MSK_SPAR_INT_SOL_FILE_NAME

Groups
Data input/output, *Solution input/output*

`sparam.itr_sol_file_name`

Name of the `itr` solution file.

Accepted
Any valid file name.

Example
```
task.putstrparam(sparam.itr_sol_file_name, "somevalue")
```

Generic name
MSK_SPAR_ITR_SOL_FILE_NAME

Groups
Data input/output, *Solution input/output*

`sparam.mio_debug_string`

For internal debugging purposes.

Accepted
Any valid string.

Example
```
task.putstrparam(sparam.mio_debug_string, "somevalue")
```

Generic name
MSK_SPAR_MIO_DEBUG_STRING

Groups
Data input/output

`sparam.param_comment_sign`

Only the first character in this string is used. It is considered as a start of comment sign in the **MOSEK** parameter file. Spaces are ignored in the string.

Default
%%

Accepted
Any valid string.

Example
```
task.putstrparam(sparam.param_comment_sign, "%%")
```

Generic name
MSK_SPAR_PARAM_COMMENT_SIGN

Groups
Data input/output

`sparam.param_read_file_name`

Modifications to the parameter database is read from this file.

Accepted
Any valid file name.

Example
```
task.putstrparam(sparam.param_read_file_name, "somevalue")
```

Generic name
MSK_SPAR_PARAM_READ_FILE_NAME

Groups
Data input/output

`sparam.param_write_file_name`

The parameter database is written to this file.

Accepted
Any valid file name.

Example
```
task.putstrparam(sparam.param_write_file_name, "somevalue")
```

Generic name
MSK_SPAR_PARAM_WRITE_FILE_NAME

Groups
Data input/output

`sparam.read_mps_bou_name`

Name of the BOUNDS vector used. An empty name means that the first BOUNDS vector is used.

Accepted
Any valid MPS name.

Example
```
task.putstrparam(sparam.read_mps_bou_name, "somevalue")
```

Generic name
MSK_SPAR_READ_MPS_BOU_NAME

Groups
Data input/output

`sparam.read_mps_obj_name`

Name of the free constraint used as objective function. An empty name means that the first constraint is used as objective function.

Accepted
Any valid MPS name.

Example
```
task.putstrparam(sparam.read_mps_obj_name, "somevalue")
```

Generic name
MSK_SPAR_READ_MPS_OBJ_NAME

Groups
Data input/output

`sparam.read_mps_ran_name`

Name of the RANGE vector used. An empty name means that the first RANGE vector is used.

Accepted
Any valid MPS name.

Example
```
task.putstrparam(sparam.read_mps_ran_name, "somevalue")
```

Generic name
MSK_SPAR_READ_MPS_RAN_NAME

Groups
Data input/output

`sparam.read_mps_rhs_name`

Name of the RHS used. An empty name means that the first RHS vector is used.

Accepted
Any valid MPS name.

Example
```
task.putstrparam(sparam.read_mps_rhs_name, "somevalue")
```

Generic name
MSK_SPAR_READ_MPS_RHS_NAME

Groups
Data input/output

`sparam.remote_optserver_host`

URL of the remote optimization server in the format `(http|https)://server:port`. If set, all subsequent calls to any **MOSEK** function that involves synchronous optimization will be sent to the specified OptServer instead of being executed locally. Passing empty string deactivates this redirection.

Accepted
Any valid URL.

Example
```
task.putstrparam(sparam.remote_optserver_host, "somevalue")
```

Generic name
MSK_SPAR_REMOTE_OPTSERVER_HOST

Groups
Overall system

`sparam.remote_tls_cert`

List of known server certificates in PEM format.

Accepted
PEM files separated by new-lines.

Example
```
task.putstrparam(sparam.remote_tls_cert, "somevalue")
```

Generic name
MSK_SPAR_REMOTE_TLS_CERT

Groups
Overall system

`sparam.remote_tls_cert_path`

Path to known server certificates in PEM format.

Accepted
Any valid path.

Example
```
task.putstrparam(sparam.remote_tls_cert_path, "somevalue")
```

Generic name
MSK_SPAR_REMOTE_TLS_CERT_PATH

Groups
Overall system

`sparam.sensitivity_file_name`

If defined *Task.sensitivityreport* reads this file as a sensitivity analysis data file specifying the type of analysis to be done.

Accepted
Any valid string.

Example
```
task.putstrparam(sparam.sensitivity_file_name, "somevalue")
```

Generic name
MSK_SPAR_SENSITIVITY_FILE_NAME

Groups
Data input/output

`sparam.sensitivity_res_file_name`

If this is a nonempty string, then *Task.sensitivityreport* writes results to this file.

Accepted
Any valid string.

Example
```
task.putstrparam(sparam.sensitivity_res_file_name, "somevalue")
```

Generic name
MSK_SPAR_SENSITIVITY_RES_FILE_NAME

Groups
Data input/output

`sparam.sol_filter_xc_low`

A filter used to determine which constraints should be listed in the solution file. A value of 0.5 means that all constraints having `xc[i]>0.5` should be listed, whereas `+0.5` means that all constraints having `xc[i]>=blc[i]+0.5` should be listed. An empty filter means that no filter is applied.

Accepted
Any valid filter.

Example
```
task.putstrparam(sparam.sol_filter_xc_low, "somevalue")
```

Generic name
MSK_SPAR_SOL_FILTER_XC_LOW

Groups
Data input/output, *Solution input/output*

`sparam.sol_filter_xc_upr`

A filter used to determine which constraints should be listed in the solution file. A value of `0.5` means that all constraints having `xc[i]<0.5` should be listed, whereas `-0.5` means all constraints having `xc[i]<=buc[i]-0.5` should be listed. An empty filter means that no filter is applied.

Accepted
Any valid filter.

Example
```
task.putstrparam(sparam.sol_filter_xc_upr, "somevalue")
```

Generic name
MSK_SPAR_SOL_FILTER_XC_UPR

Groups
Data input/output, *Solution input/output*

`sparam.sol_filter_xx_low`

A filter used to determine which variables should be listed in the solution file. A value of "0.5" means that all constraints having `xx[j]>=0.5` should be listed, whereas "+0.5" means that all constraints having `xx[j]>=blx[j]+0.5` should be listed. An empty filter means no filter is applied.

Accepted
Any valid filter.

Example
```
task.putstrparam(sparam.sol_filter_xx_low, "somevalue")
```

Generic name
MSK_SPAR_SOL_FILTER_XX_LOW

Groups
Data input/output, *Solution input/output*

`sparam.sol_filter_xx_upr`

A filter used to determine which variables should be listed in the solution file. A value of "0.5" means that all constraints having `xx[j]<0.5` should be printed, whereas "-0.5" means all constraints having `xx[j]<=bux[j]-0.5` should be listed. An empty filter means no filter is applied.

Accepted
Any valid file name.

Example
```
task.putstrparam(sparam.sol_filter_xx_upr, "somevalue")
```

Generic name
MSK_SPAR_SOL_FILTER_XX_UPR

Groups
Data input/output, *Solution input/output*

`sparam.stat_key`

Key used when writing the summary file.

Accepted
Any valid string.

Example
```
task.putstrparam(sparam.stat_key, "somevalue")
```

Generic name
MSK_SPAR_STAT_KEY

Groups
Data input/output

`sparam.stat_name`

Name used when writing the statistics file.

Accepted
Any valid XML string.

Example
```
task.putstrparam(sparam.stat_name, "somevalue")
```

Generic name
MSK_SPAR_STAT_NAME

Groups
Data input/output

15.8 Response codes

Response codes include:

- *Termination codes*
- *Warnings*
- *Errors*

The numerical code (in brackets) identifies the response in error messages and in the log output.

`rescode`

The enumeration type containing all response codes.

15.8.1 Termination

`rescode.ok (0)`

No error occurred.

`rescode.trm_max_iterations (100000)`

The optimizer terminated at the maximum number of iterations.

`rescode.trm_max_time (100001)`

The optimizer terminated at the maximum amount of time.

`rescode.trm_objective_range (100002)`

The optimizer terminated with an objective value outside the objective range.

`rescode.trm_mio_num_relaxs (100008)`

The mixed-integer optimizer terminated as the maximum number of relaxations was reached.

`rescode.trm_mio_num_branches (100009)`

The mixed-integer optimizer terminated as the maximum number of branches was reached.

`rescode.trm_num_max_num_int_solutions (100015)`

The mixed-integer optimizer terminated as the maximum number of feasible solutions was reached.

`rescode.trm_stall (100006)`

The optimizer is terminated due to slow progress.

Stalling means that numerical problems prevent the optimizer from making reasonable progress and that it makes no sense to continue. In many cases this happens if the problem is badly scaled or otherwise ill-conditioned. There is no guarantee that the solution will be feasible or optimal. However, often stalling happens near the optimum, and the returned solution may be of good quality. Therefore, it is recommended to check the status of the solution. If the solution status is optimal the solution is most likely good enough for most practical purposes.

Please note that if a linear optimization problem is solved using the interior-point optimizer with basis identification turned on, the returned basic solution likely to have high accuracy, even though the optimizer stalled.

Some common causes of stalling are a) badly scaled models, b) near feasible or near infeasible problems.

`rescode.trm_user_callback (100007)`

The optimizer terminated due to the return of the user-defined callback function.

`rescode.trm_max_num_setbacks (100020)`

The optimizer terminated as the maximum number of set-backs was reached. This indicates serious numerical problems and a possibly badly formulated problem.

`rescode.trm_numerical_problem (100025)`

The optimizer terminated due to numerical problems.

`rescode.trm_lost_race (100027)`

Lost a race.

`rescode.trm_internal (100030)`

The optimizer terminated due to some internal reason. Please contact **MOSEK** support.

`rescode.trm_internal_stop (100031)`

The optimizer terminated for internal reasons. Please contact **MOSEK** support.

`rescode.trm_server_max_time (100032)`

remote server terminated **MOSEK** on time limit criteria.

`rescode.trm_server_max_memory (100033)`

remote server terminated **MOSEK** on memory limit criteria.

15.8.2 Warnings

`rescode.wrn_open_param_file (50)`

The parameter file could not be opened.

`rescode.wrn_large_bound (51)`

A numerically large bound value is specified.

`rescode.wrn_large_lo_bound (52)`

A numerically large lower bound value is specified.

`rescode.wrn_large_up_bound (53)`

A numerically large upper bound value is specified.

`rescode.wrn_large_con_fx (54)`

An equality constraint is fixed to a numerically large value. This can cause numerical problems.

`rescode.wrn_large_cj (57)`

A numerically large value is specified for one c_j.

`rescode.wrn_large_aij (62)`

A numerically large value is specified for an $a_{i,j}$ element in A. The parameter *dparam.data_tol_aij_large* controls when an $a_{i,j}$ is considered large.

`rescode.wrn_zero_aij (63)`

One or more zero elements are specified in A.

`rescode.wrn_name_max_len (65)`

A name is longer than the buffer that is supposed to hold it.

`rescode.wrn_spar_max_len (66)`

A value for a string parameter is longer than the buffer that is supposed to hold it.

`rescode.wrn_mps_split_rhs_vector (70)`

An RHS vector is split into several nonadjacent parts in an MPS file.

`rescode.wrn_mps_split_ran_vector (71)`

A RANGE vector is split into several nonadjacent parts in an MPS file.

`rescode.wrn_mps_split_bou_vector (72)`

A BOUNDS vector is split into several nonadjacent parts in an MPS file.

`rescode.wrn_lp_old_quad_format (80)`

Missing '/2' after quadratic expressions in bound or objective.

`rescode.wrn_lp_drop_variable (85)`

Ignored a variable because the variable was not previously defined. Usually this implies that a variable appears in the bound section but not in the objective or the constraints.

`rescode.wrn_nz_in_upr_tri (200)`

Non-zero elements specified in the upper triangle of a matrix were ignored.

`rescode.wrn_dropped_nz_qobj (201)`

One or more non-zero elements were dropped in the Q matrix in the objective.

`rescode.wrn_ignore_integer (250)`

Ignored integer constraints.

`rescode.wrn_no_global_optimizer (251)`

No global optimizer is available.

`rescode.wrn_mio_infeasible_final (270)`

The final mixed-integer problem with all the integer variables fixed at their optimal values is infeasible.

`rescode.wrn_sol_filter (300)`

Invalid solution filter is specified.

`rescode.wrn_undef_sol_file_name (350)`

Undefined name occurred in a solution.

`rescode.wrn_sol_file_ignored_con (351)`

One or more lines in the constraint section were ignored when reading a solution file.

`rescode.wrn_sol_file_ignored_var (352)`

One or more lines in the variable section were ignored when reading a solution file.

`rescode.wrn_too_few_basis_vars (400)`

An incomplete basis has been specified. Too few basis variables are specified.

`rescode.wrn_too_many_basis_vars (405)`

A basis with too many variables has been specified.

`rescode.wrn_license_expire (500)`

The license expires.

`rescode.wrn_license_server (501)`

The license server is not responding.

`rescode.wrn_empty_name (502)`

A variable or constraint name is empty. The output file may be invalid.

`rescode.wrn_using_generic_names (503)`

Generic names are used because a name invalid. For instance when writing an LP file the names must not contain blanks or start with a digit. Also remeber to give the objective function a name.

`rescode.wrn_invalid_mps_name (504)`

A name e.g. a row name is not a valid MPS name.

`rescode.wrn_invalid_mps_obj_name (505)`

The objective name is not a valid MPS name.

`rescode.wrn_license_feature_expire (509)`

The license expires.

`rescode.wrn_param_name_dou (510)`

The parameter name is not recognized as a double parameter.

`rescode.wrn_param_name_int (511)`

The parameter name is not recognized as a integer parameter.

`rescode.wrn_param_name_str (512)`

The parameter name is not recognized as a string parameter.

`rescode.wrn_param_str_value (515)`

The string is not recognized as a symbolic value for the parameter.

`rescode.wrn_param_ignored_cmio (516)`

A parameter was ignored by the conic mixed integer optimizer.

`rescode.wrn_zeros_in_sparse_row (705)`

One or more (near) zero elements are specified in a sparse row of a matrix. Since, it is redundant to specify zero elements then it may indicate an error.

`rescode.wrn_zeros_in_sparse_col (710)`

One or more (near) zero elements are specified in a sparse column of a matrix. It is redundant to specify zero elements. Hence, it may indicate an error.

`rescode.wrn_incomplete_linear_dependency_check (800)`

The linear dependency check(s) is incomplete. Normally this is not an important warning unless the optimization problem has been formulated with linear dependencies. Linear dependencies may prevent **MOSEK** from solving the problem.

`rescode.wrn_eliminator_space (801)`

The eliminator is skipped at least once due to lack of space.

`rescode.wrn_presolve_outofspace (802)`

The presolve is incomplete due to lack of space.

`rescode.wrn_presolve_primal_perturbations (803)`

The presolve perturbed the bounds of the primal problem. This is an indication that the problem is nearly infeasible.

`rescode.wrn_write_changed_names (830)`

Some names were changed because they were invalid for the output file format.

`rescode.wrn_write_discarded_cfix (831)`

The fixed objective term could not be converted to a variable and was discarded in the output file.

`rescode.wrn_duplicate_constraint_names (850)`

Two constraint names are identical.

`rescode.wrn_duplicate_variable_names (851)`

Two variable names are identical.

`rescode.wrn_duplicate_barvariable_names (852)`

Two barvariable names are identical.

`rescode.wrn_duplicate_cone_names (853)`

Two cone names are identical.

`rescode.wrn_ana_large_bounds (900)`

This warning is issued by the problem analyzer, if one or more constraint or variable bounds are very large. One should consider omitting these bounds entirely by setting them to +inf or -inf.

`rescode.wrn_ana_c_zero (901)`

This warning is issued by the problem analyzer, if the coefficients in the linear part of the objective are all zero.

`rescode.wrn_ana_empty_cols (902)`

This warning is issued by the problem analyzer, if columns, in which all coefficients are zero, are found.

`rescode.wrn_ana_close_bounds (903)`

This warning is issued by problem analyzer, if ranged constraints or variables with very close upper and lower bounds are detected. One should consider treating such constraints as equalities and such variables as constants.

`rescode.wrn_ana_almost_int_bounds (904)`

This warning is issued by the problem analyzer if a constraint is bound nearly integral.

`rescode.wrn_no_infeasibility_report_when_matrix_variables (930)`

An infeasibility report is not available when the problem contains matrix variables.

`rescode.wrn_getdual_ignores_integrality (940)`

Dualizer ignores integer variables and disjunctive constraints.

`rescode.wrn_no_dualizer (950)`

No automatic dualizer is available for the specified problem. The primal problem is solved.

`rescode.wrn_sym_mat_large (960)`

A numerically large value is specified for an $e_{i,j}$ element in E. The parameter *dparam.data_sym_mat_tol_large* controls when an $e_{i,j}$ is considered large.

`rescode.wrn_modified_double_parameter (970)`

A double parameter related to solver tolerances has a non-default value.

`rescode.wrn_large_fij (980)`

A numerically large value is specified for an $f_{i,j}$ element in F. The parameter *dparam.data_tol_aij_large* controls when an $f_{i,j}$ is considered large.

`rescode.wrn_ptf_unknown_section (981)`

Unexpected section in PTF file

15.8.3 Errors

`rescode.err_license (1000)`

Invalid license.

`rescode.err_license_expired (1001)`

The license has expired.

`rescode.err_license_version (1002)`

The license is valid for another version of **MOSEK**.

`rescode.err_license_old_server_version (1003)`

The version of the FlexLM license server is too old. You should upgrade the license server to one matching this version of **MOSEK**. It will support this and all older versions of **MOSEK**.

This error can appear if the client was updated to a new version which includes an upgrade of the licensing module, making it incompatible with a much older license server.

`rescode.err_size_license (1005)`

The problem is bigger than the license.

`rescode.err_prob_license (1006)`

The software is not licensed to solve the problem.

`rescode.err_file_license (1007)`

Invalid license file.

`rescode.err_missing_license_file (1008)`

MOSEK cannot find license file or a token server. See the **MOSEK** licensing manual for details.

`rescode.err_size_license_con (1010)`

The problem has too many constraints to be solved with the available license.

`rescode.err_size_license_var (1011)`

The problem has too many variables to be solved with the available license.

`rescode.err_size_license_intvar (1012)`

The problem contains too many integer variables to be solved with the available license.

`rescode.err_optimizer_license (1013)`

The optimizer required is not licensed.

`rescode.err_flexlm (1014)`

The FLEXlm license manager reported an error.

`rescode.err_license_server (1015)`

The license server is not responding.

`rescode.err_license_max (1016)`

Maximum number of licenses is reached.

`rescode.err_license_moseklm_daemon (1017)`

The MOSEKLM license manager daemon is not up and running.

`rescode.err_license_feature (1018)`

A requested feature is not available in the license file(s). Most likely due to an incorrect license system setup.

`rescode.err_platform_not_licensed (1019)`

A requested license feature is not available for the required platform.

`rescode.err_license_cannot_allocate (1020)`

The license system cannot allocate the memory required.

`rescode.err_license_cannot_connect (1021)`

MOSEK cannot connect to the license server. Most likely the license server is not up and running.

`rescode.err_license_invalid_hostid (1025)`

The host ID specified in the license file does not match the host ID of the computer.

`rescode.err_license_server_version (1026)`

The version specified in the checkout request is greater than the highest version number the daemon supports.

`rescode.err_license_no_server_support (1027)`

The license server does not support the requested feature. Possible reasons for this error include:

- The feature has expired.
- The feature's start date is later than today's date.
- The version requested is higher than feature's the highest supported version.
- A corrupted license file.

Try restarting the license and inspect the license server debug file, usually called `lmgrd.log`.

`rescode.err_license_no_server_line (1028)`

There is no `SERVER` line in the license file. All non-zero license count features need at least one `SERVER` line.

`rescode.err_older_dll (1035)`

The dynamic link library is older than the specified version.

`rescode.err_newer_dll (1036)`

The dynamic link library is newer than the specified version.

`rescode.err_link_file_dll (1040)`

A file cannot be linked to a stream in the DLL version.

`rescode.err_thread_mutex_init (1045)`

Could not initialize a mutex.

`rescode.err_thread_mutex_lock (1046)`

Could not lock a mutex.

`rescode.err_thread_mutex_unlock (1047)`

Could not unlock a mutex.

`rescode.err_thread_create (1048)`

Could not create a thread. This error may occur if a large number of environments are created and not deleted again. In any case it is a good practice to minimize the number of environments created.

`rescode.err_thread_cond_init (1049)`

Could not initialize a condition.

`rescode.err_unknown (1050)`

Unknown error.

`rescode.err_space (1051)`

Out of space.

`rescode.err_file_open (1052)`

Error while opening a file.

`rescode.err_file_read (1053)`

File read error.

`rescode.err_file_write (1054)`

File write error.

`rescode.err_data_file_ext (1055)`

The data file format cannot be determined from the file name.

`rescode.err_invalid_file_name (1056)`

An invalid file name has been specified.

`rescode.err_invalid_sol_file_name (1057)`

An invalid file name has been specified.

`rescode.err_end_of_file (1059)`

End of file has been reached unexpectedly.

`rescode.err_null_env (1060)`

`env` is a `None` pointer.

`rescode.err_null_task (1061)`

`task` is a `None` pointer.

`rescode.err_invalid_stream (1062)`

An invalid stream is referenced.

`rescode.err_no_init_env (1063)`

`env` is not initialized.

`rescode.err_invalid_task (1064)`

The `task` is invalid.

`rescode.err_null_pointer (1065)`

An argument to a function is unexpectedly a `None` pointer.

`rescode.err_living_tasks (1066)`

All tasks associated with an enviroment must be deleted before the environment is deleted. There are still some undeleted tasks.

`rescode.err_read_gzip (1067)`

Error encountered in GZIP stream.

`rescode.err_read_zstd (1068)`

Error encountered in ZSTD stream.

`rescode.err_read_async (1069)`

Error encountered in async stream.

`rescode.err_blank_name (1070)`

An all blank name has been specified.

`rescode.err_dup_name (1071)`

The same name was used multiple times for the same problem item type.

`rescode.err_format_string (1072)`

The name format string is invalid.

`rescode.err_sparsity_specification (1073)`

The sparsity included an index that was out of bounds of the shape.

`rescode.err_mismatching_dimension (1074)`

Mismatching dimensions specified in arguments

`rescode.err_invalid_obj_name (1075)`

An invalid objective name is specified.

`rescode.err_invalid_con_name (1076)`

An invalid constraint name is used.

`rescode.err_invalid_var_name (1077)`

An invalid variable name is used.

`rescode.err_invalid_cone_name (1078)`

An invalid cone name is used.

`rescode.err_invalid_barvar_name (1079)`

An invalid symmetric matrix variable name is used.

`rescode.err_space_leaking (1080)`

MOSEK is leaking memory. This can be due to either an incorrect use of **MOSEK** or a bug.

`rescode.err_space_no_info (1081)`

No available information about the space usage.

`rescode.err_dimension_specification (1082)`

Invalid dimension specification

`rescode.err_axis_name_specification (1083)`

Invalid axis names specification

`rescode.err_read_premature_eof (1089)`

Encountered premature end-of-file in input stream.

`rescode.err_read_format (1090)`

The specified format cannot be read.

`rescode.err_write_lp_invalid_var_names (1091)`

Invalid variable name. Cannot write valid LP file.

`rescode.err_write_lp_duplicate_var_names (1092)`

Duplicate variable names. Cannot write valid LP file.

`rescode.err_write_lp_invalid_con_names (1093)`

Invalid constraint name. Cannot write valid LP file.

`rescode.err_write_lp_duplicate_con_names (1094)`

Duplicate constraint names. Cannot write valid LP file.

`rescode.err_mps_file (1100)`

An error occurred while reading an MPS file.

`rescode.err_mps_inv_field (1101)`

A field in the MPS file is invalid. Probably it is too wide.

`rescode.err_mps_inv_marker (1102)`

An invalid marker has been specified in the MPS file.

`rescode.err_mps_null_con_name (1103)`

An empty constraint name is used in an MPS file.

`rescode.err_mps_null_var_name (1104)`

An empty variable name is used in an MPS file.

`rescode.err_mps_undef_con_name (1105)`

An undefined constraint name occurred in an MPS file.

`rescode.err_mps_undef_var_name (1106)`

An undefined variable name occurred in an MPS file.

`rescode.err_mps_invalid_con_key (1107)`

An invalid constraint key occurred in an MPS file.

`rescode.err_mps_invalid_bound_key (1108)`

An invalid bound key occurred in an MPS file.

`rescode.err_mps_invalid_sec_name (1109)`

An invalid section name occurred in an MPS file.

`rescode.err_mps_no_objective (1110)`

No objective is defined in an MPS file.

`rescode.err_mps_splitted_var (1111)`

All elements in a column of the A matrix must be specified consecutively. Hence, it is illegal to specify non-zero elements in A for variable 1, then for variable 2 and then variable 1 again.

`rescode.err_mps_mul_con_name (1112)`

A constraint name was specified multiple times in the `ROWS` section.

`rescode.err_mps_mul_qsec (1113)`

Multiple `QSECTION`s are specified for a constraint in the MPS data file.

`rescode.err_mps_mul_qobj (1114)`

The Q term in the objective is specified multiple times in the MPS data file.

`rescode.err_mps_inv_sec_order (1115)`

The sections in the MPS data file are not in the correct order.

`rescode.err_mps_mul_csec (1116)`

Multiple `CSECTION`s are given the same name.

`rescode.err_mps_cone_type (1117)`

Invalid cone type specified in a `CSECTION`.

`rescode.err_mps_cone_overlap (1118)`

A variable is specified to be a member of several cones.

`rescode.err_mps_cone_repeat (1119)`

A variable is repeated within the `CSECTION`.

`rescode.err_mps_non_symmetric_q (1120)`

A non symmetric matrix has been speciefied.

`rescode.err_mps_duplicate_q_element (1121)`

Duplicate elements is specfied in a Q matrix.

`rescode.err_mps_invalid_objsense (1122)`

An invalid objective sense is specified.

`rescode.err_mps_tab_in_field2 (1125)`

A tab char occurred in field 2.

`rescode.err_mps_tab_in_field3 (1126)`

A tab char occurred in field 3.

`rescode.err_mps_tab_in_field5 (1127)`

A tab char occurred in field 5.

`rescode.err_mps_invalid_obj_name (1128)`

An invalid objective name is specified.

`rescode.err_mps_invalid_key (1129)`

An invalid indicator key occurred in an MPS file.

`rescode.err_mps_invalid_indicator_constraint (1130)`

An invalid indicator constraint is used. It must not be a ranged constraint.

`rescode.err_mps_invalid_indicator_variable (1131)`

An invalid indicator variable is specfied. It must be a binary variable.

`rescode.err_mps_invalid_indicator_value (1132)`

An invalid indicator value is specfied. It must be either 0 or 1.

`rescode.err_mps_invalid_indicator_quadratic_constraint (1133)`

A quadratic constraint can be be an indicator constraint.

`rescode.err_opf_syntax (1134)`

Syntax error in an OPF file

`rescode.err_opf_premature_eof (1136)`

Premature end of file in an OPF file.

`rescode.err_opf_mismatched_tag (1137)`

Mismatched end-tag in OPF file

`rescode.err_opf_duplicate_bound (1138)`

Either upper or lower bound was specified twice in OPF file

`rescode.err_opf_duplicate_constraint_name (1139)`

Duplicate constraint name in OPF File

`rescode.err_opf_invalid_cone_type (1140)`

Invalid cone type in OPF File

`rescode.err_opf_incorrect_tag_param (1141)`

Invalid number of parameters in start-tag in OPF File

`rescode.err_opf_invalid_tag (1142)`

Invalid start-tag in OPF File

`rescode.err_opf_duplicate_cone_entry (1143)`

Same variable appears in multiple cones in OPF File

`rescode.err_opf_too_large (1144)`

The problem is too large to be correctly loaded

`rescode.err_opf_dual_integer_solution (1146)`

Dual solution values are not allowed in OPF File

`rescode.err_lp_empty (1151)`

The problem cannot be written to an LP formatted file.

`rescode.err_write_mps_invalid_name (1153)`

An invalid name is created while writing an MPS file. Usually this will make the MPS file unreadable.

`rescode.err_lp_invalid_var_name (1154)`

A variable name is invalid when used in an LP formatted file.

`rescode.err_write_opf_invalid_var_name (1156)`

Empty variable names cannot be written to OPF files.

`rescode.err_lp_file_format (1157)`

Syntax error in an LP file.

`rescode.err_lp_expected_number (1158)`

Expected a number in LP file

`rescode.err_read_lp_missing_end_tag (1159)`

Syntax error in LP file. Possibly missing End tag.

`rescode.err_lp_indicator_var (1160)`

An indicator variable was not declared binary

`rescode.err_lp_expected_objective (1161)`

Expected an objective section in LP file

`rescode.err_lp_expected_constraint_relation (1162)`

Expected constraint relation

`rescode.err_lp_ambiguous_constraint_bound (1163)`

Constraint has ambiguous or invalid bound

`rescode.err_lp_duplicate_section (1164)`

Duplicate section

`rescode.err_read_lp_delayed_rows_not_supported (1165)`

Duplicate section

`rescode.err_writing_file (1166)`

An error occurred while writing file

`rescode.err_write_async (1167)`

An error occurred while performing asynchronous writing

`rescode.err_invalid_name_in_sol_file (1170)`

An invalid name occurred in a solution file.

`rescode.err_json_syntax (1175)`

Syntax error in an JSON data

`rescode.err_json_string (1176)`

Error in JSON string.

`rescode.err_json_number_overflow (1177)`

Invalid number entry - wrong type or value overflow.

`rescode.err_json_format (1178)`

Error in an JSON Task file

`rescode.err_json_data (1179)`

Inconsistent data in JSON Task file

`rescode.err_json_missing_data (1180)`

Missing data section in JSON task file.

`rescode.err_ptf_incompatibility (1181)`

Incompatible item

`rescode.err_ptf_undefined_item (1182)`

Undefined symbol referenced

`rescode.err_ptf_inconsistency (1183)`

Inconsistent size of item

`rescode.err_ptf_format (1184)`

Syntax error in an PTF file

`rescode.err_argument_lenneq (1197)`

Incorrect length of arguments.

`rescode.err_argument_type (1198)`

Incorrect argument type.

`rescode.err_num_arguments (1199)`

Incorrect number of function arguments.

`rescode.err_in_argument (1200)`

A function argument is incorrect.

`rescode.err_argument_dimension (1201)`

A function argument is of incorrect dimension.

`rescode.err_shape_is_too_large (1202)`

The size of the n-dimensional shape is too large.

`rescode.err_index_is_too_small (1203)`

An index in an argument is too small.

`rescode.err_index_is_too_large (1204)`

An index in an argument is too large.

`rescode.err_index_is_not_unique (1205)`

An index in an argument is not unique.

`rescode.err_param_name (1206)`

The parameter name is not correct.

`rescode.err_param_name_dou (1207)`

The parameter name is not correct for a double parameter.

`rescode.err_param_name_int (1208)`

The parameter name is not correct for an integer parameter.

`rescode.err_param_name_str (1209)`

The parameter name is not correct for a string parameter.

`rescode.err_param_index (1210)`

Parameter index is out of range.

`rescode.err_param_is_too_large (1215)`

The parameter value is too large.

`rescode.err_param_is_too_small (1216)`

The parameter value is too small.

`rescode.err_param_value_str (1217)`

The parameter value string is incorrect.

`rescode.err_param_type (1218)`

The parameter type is invalid.

`rescode.err_inf_dou_index (1219)`

A double information index is out of range for the specified type.

`rescode.err_inf_int_index (1220)`

An integer information index is out of range for the specified type.

`rescode.err_index_arr_is_too_small (1221)`

An index in an array argument is too small.

`rescode.err_index_arr_is_too_large (1222)`

An index in an array argument is too large.

`rescode.err_inf_lint_index (1225)`

A long integer information index is out of range for the specified type.

`rescode.err_arg_is_too_small (1226)`

The value of a argument is too small.

`rescode.err_arg_is_too_large (1227)`

The value of a argument is too large.

`rescode.err_invalid_whichsol (1228)`

`whichsol` is invalid.

`rescode.err_inf_dou_name (1230)`

A double information name is invalid.

`rescode.err_inf_int_name (1231)`

An integer information name is invalid.

`rescode.err_inf_type (1232)`

The information type is invalid.

`rescode.err_inf_lint_name (1234)`

A long integer information name is invalid.

`rescode.err_index (1235)`

An index is out of range.

`rescode.err_whichsol (1236)`

The solution defined by `whichsol` does not exists.

`rescode.err_solitem (1237)`

The solution item number `solitem` is invalid. Please note that *solitem.snx* is invalid for the basic solution.

`rescode.err_whichitem_not_allowed (1238)`

`whichitem` is unacceptable.

`rescode.err_maxnumcon (1240)`

The maximum number of constraints specified is smaller than the number of constraints in the task.

`rescode.err_maxnumvar (1241)`

The maximum number of variables specified is smaller than the number of variables in the task.

`rescode.err_maxnumbarvar (1242)`

The maximum number of semidefinite variables specified is smaller than the number of semidefinite variables in the task.

`rescode.err_maxnumqnz (1243)`

The maximum number of non-zeros specified for the Q matrices is smaller than the number of non-zeros in the current Q matrices.

`rescode.err_too_small_max_num_nz (1245)`

The maximum number of non-zeros specified is too small.

`rescode.err_invalid_idx (1246)`

A specified index is invalid.

`rescode.err_invalid_max_num (1247)`

A specified index is invalid.

`rescode.err_unallowed_whichsol (1248)`

The value od `whichsol` is not allowed.

`rescode.err_numconlim (1250)`

Maximum number of constraints limit is exceeded.

`rescode.err_numvarlim (1251)`

Maximum number of variables limit is exceeded.

`rescode.err_too_small_maxnumanz (1252)`

The maximum number of non-zeros specified for A is smaller than the number of non-zeros in the current A.

`rescode.err_inv_aptre (1253)`

`aptre[j]` is strictly smaller than `aptrb[j]` for some `j`.

`rescode.err_mul_a_element (1254)`

An element in A is defined multiple times.

`rescode.err_inv_bk (1255)`

Invalid bound key.

`rescode.err_inv_bkc (1256)`

Invalid bound key is specified for a constraint.

`rescode.err_inv_bkx (1257)`

An invalid bound key is specified for a variable.

`rescode.err_inv_var_type (1258)`

An invalid variable type is specified for a variable.

`rescode.err_solver_probtype (1259)`

Problem type does not match the chosen optimizer.

`rescode.err_objective_range (1260)`

Empty objective range.

`rescode.err_inv_rescode (1261)`

Invalid response code.

`rescode.err_inv_iinf (1262)`

Invalid integer information item.

`rescode.err_inv_liinf (1263)`

Invalid long integer information item.

`rescode.err_inv_dinf (1264)`

Invalid double information item.

`rescode.err_basis (1266)`

An invalid basis is specified. Either too many or too few basis variables are specified.

`rescode.err_inv_skc (1267)`

Invalid value in `skc`.

`rescode.err_inv_skx (1268)`

Invalid value in `skx`.

`rescode.err_inv_skn (1274)`

Invalid value in `skn`.

`rescode.err_inv_sk_str (1269)`

Invalid status key string encountered.

`rescode.err_inv_sk (1270)`

Invalid status key code.

`rescode.err_inv_cone_type_str (1271)`

Invalid cone type string encountered.

`rescode.err_inv_cone_type (1272)`

Invalid cone type code is encountered.

`rescode.err_invalid_surplus (1275)`

Invalid surplus.

`rescode.err_inv_name_item (1280)`

An invalid name item code is used.

`rescode.err_pro_item (1281)`

An invalid problem is used.

`rescode.err_invalid_format_type (1283)`

Invalid format type.

`rescode.err_firsti (1285)`

Invalid `firsti`.

`rescode.err_lasti (1286)`

Invalid `lasti`.

`rescode.err_firstj (1287)`

Invalid `firstj`.

`rescode.err_lastj (1288)`

Invalid `lastj`.

`rescode.err_max_len_is_too_small (1289)`

A maximum length that is too small has been specified.

`rescode.err_nonlinear_equality (1290)`

The model contains a nonlinear equality which defines a nonconvex set.

`rescode.err_nonconvex (1291)`

The optimization problem is nonconvex.

`rescode.err_nonlinear_ranged (1292)`

Nonlinear constraints with finite lower and upper bound always define a nonconvex feasible set.

`rescode.err_con_q_not_psd (1293)`

The quadratic constraint matrix is not positive semidefinite as expected for a constraint with finite upper bound. This results in a nonconvex problem.

`rescode.err_con_q_not_nsd (1294)`

The quadratic constraint matrix is not negative semidefinite as expected for a constraint with finite lower bound. This results in a nonconvex problem.

`rescode.err_obj_q_not_psd (1295)`

The quadratic coefficient matrix in the objective is not positive semidefinite as expected for a minimization problem.

`rescode.err_obj_q_not_nsd (1296)`

The quadratic coefficient matrix in the objective is not negative semidefinite as expected for a maximization problem.

`rescode.err_argument_perm_array (1299)`

An invalid permutation array is specified.

`rescode.err_cone_index (1300)`

An index of a non-existing cone has been specified.

`rescode.err_cone_size (1301)`

A cone with incorrect number of members is specified.

`rescode.err_cone_overlap (1302)`

One or more of the variables in the cone to be added is already member of another cone. Now assume the variable is x_j then add a new variable say x_k and the constraint

$$x_j = x_k$$

and then let x_k be member of the cone to be appended.

`rescode.err_cone_rep_var (1303)`

A variable is included multiple times in the cone.

`rescode.err_maxnumcone (1304)`

The value specified for `maxnumcone` is too small.

`rescode.err_cone_type (1305)`

Invalid cone type specified.

`rescode.err_cone_type_str (1306)`

Invalid cone type specified.

`rescode.err_cone_overlap_append (1307)`

The cone to be appended has one variable which is already member of another cone.

`rescode.err_remove_cone_variable (1310)`

A variable cannot be removed because it will make a cone invalid.

`rescode.err_appending_too_big_cone (1311)`

Trying to append a too big cone.

`rescode.err_cone_parameter (1320)`

An invalid cone parameter.

`rescode.err_sol_file_invalid_number (1350)`

An invalid number is specified in a solution file.

`rescode.err_huge_c (1375)`

A huge value in absolute size is specified for one c_j.

`rescode.err_huge_aij (1380)`

A numerically huge value is specified for an $a_{i,j}$ element in A. The parameter *dparam.data_tol_aij_huge* controls when an $a_{i,j}$ is considered huge.

`rescode.err_duplicate_aij (1385)`

An element in the A matrix is specified twice.

`rescode.err_lower_bound_is_a_nan (1390)`

The lower bound specified is not a number (nan) or is not finite.

`rescode.err_upper_bound_is_a_nan (1391)`

The upper bound specified is not a number (nan) or is not finite.

`rescode.err_infinite_bound (1400)`

A numerically huge bound value is specified.

`rescode.err_inv_qobj_subi (1401)`

Invalid value in `qosubi`.

`rescode.err_inv_qobj_subj (1402)`

Invalid value in `qosubj`.

`rescode.err_inv_qobj_val (1403)`

Invalid value in `qoval`.

`rescode.err_inv_qcon_subk (1404)`

Invalid value in `qcsubk`.

`rescode.err_inv_qcon_subi (1405)`

Invalid value in `qcsubi`.

`rescode.err_inv_qcon_subj (1406)`

Invalid value in `qcsubj`.

`rescode.err_inv_qcon_val (1407)`

Invalid value in `qcval`.

`rescode.err_qcon_subi_too_small (1408)`

Invalid value in `qcsubi`.

`rescode.err_qcon_subi_too_large (1409)`

Invalid value in `qcsubi`.

`rescode.err_qobj_upper_triangle (1415)`

An element in the upper triangle of Q^o is specified. Only elements in the lower triangle should be specified.

`rescode.err_qcon_upper_triangle (1417)`

An element in the upper triangle of a Q^k is specified. Only elements in the lower triangle should be specified.

`rescode.err_fixed_bound_values (1420)`

A fixed constraint/variable has been specified using the bound keys but the numerical value of the lower and upper bound is different.

`rescode.err_too_small_a_truncation_value (1421)`

A too small value for the A trucation value is specified.

`rescode.err_invalid_objective_sense (1445)`

An invalid objective sense is specified.

`rescode.err_undefined_objective_sense (1446)`

The objective sense has not been specified before the optimization.

`rescode.err_y_is_undefined (1449)`

The solution item y is undefined.

`rescode.err_nan_in_double_data (1450)`

An invalid floating point value was used in some double data.

`rescode.err_inf_in_double_data (1451)`

An infinite floating point value was used in some double data.

`rescode.err_nan_in_blc (1461)`

l^c contains an invalid floating point value, i.e. a `NaN` or `Inf`.

`rescode.err_nan_in_buc (1462)`

u^c contains an invalid floating point value, i.e. a `NaN` of `Inf`.

`rescode.err_invalid_cfix (1469)`

An invalid fixed term in the objective is speficied.

`rescode.err_nan_in_c (1470)`

c contains an invalid floating point value, i.e. a `NaN` or `Inf`.

`rescode.err_nan_in_blx (1471)`

l^x contains an invalid floating point value, i.e. a `NaN` or `Inf`.

`rescode.err_nan_in_bux (1472)`

u^x contains an invalid floating point value, i.e. a `NaN` or `Inf`.

`rescode.err_invalid_aij (1473)`

$a_{i,j}$ contains an invalid floating point value, i.e. a `NaN` or an infinite value.

`rescode.err_invalid_cj (1474)`

c_j contains an invalid floating point value, i.e. a `NaN` or an infinite value.

`rescode.err_sym_mat_invalid (1480)`

A symmetric matrix contains an invalid floating point value, i.e. a `NaN` or an infinite value.

`rescode.err_sym_mat_huge (1482)`

A symmetric matrix contains a huge value in absolute size. The parameter *dparam.data_sym_mat_tol_huge* controls when an $e_{i,j}$ is considered huge.

`rescode.err_inv_problem (1500)`

Invalid problem type. Probably a nonconvex problem has been specified.

`rescode.err_mixed_conic_and_nl (1501)`

The problem contains nonlinear terms conic constraints. The requested operation cannot be applied to this type of problem.

`rescode.err_global_inv_conic_problem (1503)`

The global optimizer can only be applied to problems without semidefinite variables.

`rescode.err_inv_optimizer (1550)`

An invalid optimizer has been chosen for the problem.

`rescode.err_mio_no_optimizer (1551)`

No optimizer is available for the current class of integer optimization problems.

`rescode.err_no_optimizer_var_type (1552)`

No optimizer is available for this class of optimization problems.

`rescode.err_final_solution (1560)`

An error occurred during the solution finalization.

`rescode.err_first (1570)`

Invalid `first`.

`rescode.err_last (1571)`

Invalid index `last`. A given index was out of expected range.

`rescode.err_slice_size (1572)`

Invalid slice size specified.

`rescode.err_negative_surplus (1573)`

Negative surplus.

`rescode.err_negative_append (1578)`

Cannot append a negative number.

`rescode.err_postsolve (1580)`

An error occurred during the postsolve. Please contact **MOSEK** support.

`rescode.err_overflow (1590)`

A computation produced an overflow i.e. a very large number.

`rescode.err_no_basis_sol (1600)`

No basic solution is defined.

`rescode.err_basis_factor (1610)`

The factorization of the basis is invalid.

`rescode.err_basis_singular (1615)`

The basis is singular and hence cannot be factored.

`rescode.err_factor (1650)`

An error occurred while factorizing a matrix.

`rescode.err_feasrepair_cannot_relax (1700)`

An optimization problem cannot be relaxed.

`rescode.err_feasrepair_solving_relaxed (1701)`

The relaxed problem could not be solved to optimality. Please consult the log file for further details.

`rescode.err_feasrepair_inconsistent_bound (1702)`

The upper bound is less than the lower bound for a variable or a constraint. Please correct this before running the feasibility repair.

`rescode.err_repair_invalid_problem (1710)`

The feasibility repair does not support the specified problem type.

`rescode.err_repair_optimization_failed (1711)`

Computation the optimal relaxation failed. The cause may have been numerical problems.

`rescode.err_name_max_len (1750)`

A name is longer than the buffer that is supposed to hold it.

`rescode.err_name_is_null (1760)`

The name buffer is a `None` pointer.

`rescode.err_invalid_compression (1800)`

Invalid compression type.

`rescode.err_invalid_iomode (1801)`

Invalid io mode.

`rescode.err_no_primal_infeas_cer (2000)`

A certificate of primal infeasibility is not available.

`rescode.err_no_dual_infeas_cer (2001)`

A certificate of infeasibility is not available.

`rescode.err_no_solution_in_callback (2500)`

The required solution is not available.

`rescode.err_inv_marki (2501)`

Invalid value in marki.

`rescode.err_inv_markj (2502)`

Invalid value in markj.

`rescode.err_inv_numi (2503)`

Invalid numi.

`rescode.err_inv_numj (2504)`

Invalid numj.

`rescode.err_task_incompatible (2560)`

The Task file is incompatible with this platform. This results from reading a file on a 32 bit platform generated on a 64 bit platform.

`rescode.err_task_invalid (2561)`

The Task file is invalid.

`rescode.err_task_write (2562)`

Failed to write the task file.

`rescode.err_read_write (2563)`

Failed to read or write due to an I/O error.

`rescode.err_task_premature_eof (2564)`

The Task file ended prematurely.

`rescode.err_lu_max_num_tries (2800)`

Could not compute the LU factors of the matrix within the maximum number of allowed tries.

`rescode.err_invalid_utf8 (2900)`

An invalid UTF8 string is encountered.

`rescode.err_invalid_wchar (2901)`

An invalid `wchar` string is encountered.

`rescode.err_no_dual_for_itg_sol (2950)`

No dual information is available for the integer solution.

`rescode.err_no_snx_for_bas_sol (2953)`

s_n^x is not available for the basis solution.

`rescode.err_internal (3000)`

An internal error occurred. Please report this problem.

`rescode.err_api_array_too_small (3001)`

An input array was too short.

`rescode.err_api_cb_connect (3002)`

Failed to connect a callback object.

`rescode.err_api_fatal_error (3005)`

An internal error occurred in the API. Please report this problem.

`rescode.err_api_internal (3999)`

An internal fatal error occurred in an interface function.

`rescode.err_sen_format (3050)`

Syntax error in sensitivity analysis file.

`rescode.err_sen_undef_name (3051)`

An undefined name was encountered in the sensitivity analysis file.

`rescode.err_sen_index_range (3052)`

Index out of range in the sensitivity analysis file.

`rescode.err_sen_bound_invalid_up (3053)`

Analysis of upper bound requested for an index, where no upper bound exists.

`rescode.err_sen_bound_invalid_lo (3054)`

Analysis of lower bound requested for an index, where no lower bound exists.

`rescode.err_sen_index_invalid (3055)`

Invalid range given in the sensitivity file.

`rescode.err_sen_invalid_regexp (3056)`

Syntax error in regexp or regexp longer than 1024.

`rescode.err_sen_solution_status (3057)`

No optimal solution found to the original problem given for sensitivity analysis.

`rescode.err_sen_numerical (3058)`

Numerical difficulties encountered performing the sensitivity analysis.

`rescode.err_sen_unhandled_problem_type (3080)`

Sensitivity analysis cannot be performed for the specified problem. Sensitivity analysis is only possible for linear problems.

`rescode.err_unb_step_size (3100)`

A step size in an optimizer was unexpectedly unbounded. For instance, if the step-size becomes unbounded in phase 1 of the simplex algorithm then an error occurs. Normally this will happen only if the problem is badly formulated. Please contact **MOSEK** support if this error occurs.

`rescode.err_identical_tasks (3101)`

Some tasks related to this function call were identical. Unique tasks were expected.

`rescode.err_ad_invalid_codelist (3102)`

The code list data was invalid.

`rescode.err_internal_test_failed (3500)`

An internal unit test function failed.

`rescode.err_int64_to_int32_cast (3800)`

A 64 bit integer could not be cast to a 32 bit integer.

`rescode.err_infeas_undefined (3910)`

The requested value is not defined for this solution type.

`rescode.err_no_barx_for_solution (3915)`

There is no $\overline{X}$ available for the solution specified. In particular note there are no $\overline{X}$ defined for the basic and integer solutions.

`rescode.err_no_bars_for_solution (3916)`

There is no $\bar{s}$ available for the solution specified. In particular note there are no $\bar{s}$ defined for the basic and integer solutions.

`rescode.err_bar_var_dim (3920)`

The dimension of a symmetric matrix variable has to be greater than 0.

`rescode.err_sym_mat_invalid_row_index (3940)`

A row index specified for sparse symmetric matrix is invalid.

`rescode.err_sym_mat_invalid_col_index (3941)`

A column index specified for sparse symmetric matrix is invalid.

`rescode.err_sym_mat_not_lower_tringular (3942)`

Only the lower triangular part of sparse symmetric matrix should be specified.

`rescode.err_sym_mat_invalid_value (3943)`

The numerical value specified in a sparse symmetric matrix is not a floating point value.

`rescode.err_sym_mat_duplicate (3944)`

A value in a symmetric matric as been specified more than once.

`rescode.err_invalid_sym_mat_dim (3950)`

A sparse symmetric matrix of invalid dimension is specified.

`rescode.err_invalid_file_format_for_sym_mat (4000)`

The file format does not support a problem with symmetric matrix variables.

`rescode.err_invalid_file_format_for_cfix (4001)`

The file format does not support a problem with nonzero fixed term in c.

`rescode.err_invalid_file_format_for_ranged_constraints (4002)`

The file format does not support a problem with ranged constraints.

`rescode.err_invalid_file_format_for_free_constraints (4003)`

The file format does not support a problem with free constraints.

`rescode.err_invalid_file_format_for_cones (4005)`

The file format does not support a problem with the simple cones (deprecated).

`rescode.err_invalid_file_format_for_quadratic_terms (4006)`

The file format does not support a problem with quadratic terms.

`rescode.err_invalid_file_format_for_nonlinear (4010)`

The file format does not support a problem with nonlinear terms.

`rescode.err_invalid_file_format_for_disjunctive_constraints (4011)`

The file format does not support a problem with disjunctive constraints.

`rescode.err_invalid_file_format_for_affine_conic_constraints (4012)`

The file format does not support a problem with affine conic constraints.

`rescode.err_duplicate_constraint_names (4500)`

Two constraint names are identical.

`rescode.err_duplicate_variable_names (4501)`

Two variable names are identical.

`rescode.err_duplicate_barvariable_names (4502)`

Two barvariable names are identical.

`rescode.err_duplicate_cone_names (4503)`

Two cone names are identical.

`rescode.err_duplicate_domain_names (4504)`

Two domain names are identical.

`rescode.err_duplicate_djc_names (4505)`

Two disjunctive constraint names are identical.

`rescode.err_non_unique_array (5000)`

An array does not contain unique elements.

`rescode.err_argument_is_too_small (5004)`

The value of a function argument is too small.

`rescode.err_argument_is_too_large (5005)`

The value of a function argument is too large.

`rescode.err_mio_internal (5010)`

A fatal error occurred in the mixed integer optimizer. Please contact **MOSEK** support.

`rescode.err_invalid_problem_type (6000)`

An invalid problem type.

`rescode.err_unhandled_solution_status (6010)`

Unhandled solution status.

`rescode.err_upper_triangle (6020)`

An element in the upper triangle of a lower triangular matrix is specified.

`rescode.err_lau_singular_matrix (7000)`

A matrix is singular.

`rescode.err_lau_not_positive_definite (7001)`

A matrix is not positive definite.

`rescode.err_lau_invalid_lower_triangular_matrix (7002)`

An invalid lower triangular matrix.

`rescode.err_lau_unknown (7005)`

An unknown error.

`rescode.err_lau_arg_m (7010)`

Invalid argument m.

`rescode.err_lau_arg_n (7011)`

Invalid argument n.

`rescode.err_lau_arg_k (7012)`

Invalid argument k.

`rescode.err_lau_arg_transa (7015)`

Invalid argument transa.

`rescode.err_lau_arg_transb (7016)`

Invalid argument transb.

`rescode.err_lau_arg_uplo (7017)`

Invalid argument uplo.

`rescode.err_lau_arg_trans (7018)`

Invalid argument trans.

`rescode.err_lau_invalid_sparse_symmetric_matrix (7019)`

An invalid sparse symmetric matrix is specfified. Note only the lower triangular part with no duplicates is specifed.

`rescode.err_cbf_parse (7100)`

An error occurred while parsing an CBF file.

`rescode.err_cbf_obj_sense (7101)`

An invalid objective sense is specified.

`rescode.err_cbf_no_variables (7102)`

No variables are specified.

`rescode.err_cbf_too_many_constraints (7103)`

Too many constraints specified.

`rescode.err_cbf_too_many_variables (7104)`

Too many variables specified.

`rescode.err_cbf_no_version_specified (7105)`

No version specified.

`rescode.err_cbf_syntax (7106)`

Invalid syntax.

`rescode.err_cbf_duplicate_obj (7107)`

Duplicate OBJ keyword.

`rescode.err_cbf_duplicate_con (7108)`

Duplicate CON keyword.

`rescode.err_cbf_duplicate_var (7110)`

Duplicate VAR keyword.

`rescode.err_cbf_duplicate_int (7111)`

Duplicate INT keyword.

`rescode.err_cbf_invalid_var_type (7112)`

Invalid variable type.

`rescode.err_cbf_invalid_con_type (7113)`

Invalid constraint type.

`rescode.err_cbf_invalid_domain_dimension (7114)`

Invalid domain dimension.

`rescode.err_cbf_duplicate_objacoord (7115)`

Duplicate index in OBJCOORD.

`rescode.err_cbf_duplicate_bcoord (7116)`

Duplicate index in BCOORD.

`rescode.err_cbf_duplicate_acoord (7117)`

Duplicate index in ACOORD.

`rescode.err_cbf_too_few_variables (7118)`

Too few variables defined.

`rescode.err_cbf_too_few_constraints (7119)`

Too few constraints defined.

`rescode.err_cbf_too_few_ints (7120)`

Too few ints are specified.

`rescode.err_cbf_too_many_ints (7121)`

Too many ints are specified.

`rescode.err_cbf_invalid_int_index (7122)`

Invalid INT index.

`rescode.err_cbf_unsupported (7123)`

Unsupported feature is present.

`rescode.err_cbf_duplicate_psdvar (7124)`

Duplicate PSDVAR keyword.

`rescode.err_cbf_invalid_psdvar_dimension (7125)`

Invalid PSDVAR dimension.

`rescode.err_cbf_too_few_psdvar (7126)`

Too few variables defined.

`rescode.err_cbf_invalid_exp_dimension (7127)`

Invalid dimension of a exponential cone.

`rescode.err_cbf_duplicate_pow_cones (7130)`

Multiple POWCONES specified.

`rescode.err_cbf_duplicate_pow_star_cones (7131)`

Multiple POW*CONES specified.

`rescode.err_cbf_invalid_power (7132)`

Invalid power specified.

`rescode.err_cbf_power_cone_is_too_long (7133)`

Power cone is too long.

`rescode.err_cbf_invalid_power_cone_index (7134)`

Invalid power cone index.

`rescode.err_cbf_invalid_power_star_cone_index (7135)`

Invalid power star cone index.

`rescode.err_cbf_unhandled_power_cone_type (7136)`

An unhandled power cone type.

`rescode.err_cbf_unhandled_power_star_cone_type (7137)`

An unhandled power star cone type.

`rescode.err_cbf_power_cone_mismatch (7138)`

The power cone does not match with it definition.

`rescode.err_cbf_power_star_cone_mismatch (7139)`

The power star cone does not match with it definition.

`rescode.err_cbf_invalid_number_of_cones (7140)`

Invalid number of cones.

`rescode.err_cbf_invalid_dimension_of_cones (7141)`

Invalid number of cones.

`rescode.err_cbf_invalid_num_objacoord (7150)`

Invalid number of OBJACOORD.

`rescode.err_cbf_invalid_num_objfcoord (7151)`

Invalid number of OBJFCOORD.

`rescode.err_cbf_invalid_num_acoord (7152)`

Invalid number of ACOORD.

`rescode.err_cbf_invalid_num_bcoord (7153)`

Invalid number of BCOORD.

`rescode.err_cbf_invalid_num_fcoord (7155)`

Invalid number of FCOORD.

`rescode.err_cbf_invalid_num_hcoord (7156)`

Invalid number of HCOORD.

`rescode.err_cbf_invalid_num_dcoord (7157)`

Invalid number of DCOORD.

`rescode.err_cbf_expected_a_keyword (7158)`

Expected a key word.

`rescode.err_cbf_invalid_num_psdcon (7200)`

Invalid number of PSDCON.

`rescode.err_cbf_duplicate_psdcon (7201)`

Duplicate CON keyword.

`rescode.err_cbf_invalid_dimension_of_psdcon (7202)`

Invalid PSDCON dimension.

`rescode.err_cbf_invalid_psdcon_index (7203)`

Invalid PSDCON index.

`rescode.err_cbf_invalid_psdcon_variable_index (7204)`

Invalid PSDCON index.

`rescode.err_cbf_invalid_psdcon_block_index (7205)`

Invalid PSDCON index.

`rescode.err_cbf_unsupported_change (7210)`

The CHANGE section is not supported.

`rescode.err_mio_invalid_root_optimizer (7700)`

An invalid root optimizer was selected for the problem type.

`rescode.err_mio_invalid_node_optimizer (7701)`

An invalid node optimizer was selected for the problem type.

`rescode.err_mps_write_cplex_invalid_cone_type (7750)`

An invalid cone type occurs when writing a CPLEX formatted MPS file.

`rescode.err_toconic_constr_q_not_psd (7800)`

The matrix defining the quadratric part of constraint is not positive semidefinite.

`rescode.err_toconic_constraint_fx (7801)`

The quadratic constraint is an equality, thus not convex.

`rescode.err_toconic_constraint_ra (7802)`

The quadratic constraint has finite lower and upper bound, and therefore it is not convex.

`rescode.err_toconic_constr_not_conic (7803)`

The constraint is not conic representable.

`rescode.err_toconic_objective_not_psd (7804)`

The matrix defining the quadratric part of the objective function is not positive semidefinite.

`rescode.err_getdual_not_available (7820)`

The simple dualizer is not available for this problem class.

`rescode.err_server_connect (8000)`

Failed to connect to remote solver server. The server string or the port string were invalid, or the server did not accept connection.

`rescode.err_server_protocol (8001)`

Unexpected message or data from solver server.

`rescode.err_server_status (8002)`

Server returned non-ok HTTP status code

`rescode.err_server_token (8003)`

The job ID specified is incorrect or invalid

`rescode.err_server_address (8004)`

Invalid address string

`rescode.err_server_certificate (8005)`

Invalid TLS certificate format or path

`rescode.err_server_tls_client (8006)`

Failed to create TLS cleint

`rescode.err_server_access_token (8007)`

Invalid access token

`rescode.err_server_problem_size (8008)`

The size of the problem exceeds the dimensions permitted by the instance of the OptServer where it was run.

`rescode.err_server_hard_timeout (8009)`

The hard timeout limit was reached on solver server, and the solver process was killed

`rescode.err_duplicate_index_in_a_sparse_matrix (20050)`

An element in a sparse matrix is specified twice.

`rescode.err_duplicate_index_in_afeidx_list (20060)`

An index is specified twice in an affine expression list.

`rescode.err_duplicate_fij (20100)`

An element in the F matrix is specified twice.

`rescode.err_invalid_fij (20101)`

$f_{i,j}$ contains an invalid floating point value, i.e. a `NaN` or an infinite value.

`rescode.err_huge_fij (20102)`

A numerically huge value is specified for an $f_{i,j}$ element in F. The parameter *dparam.data_tol_aij_huge* controls when an $f_{i,j}$ is considered huge.

`rescode.err_invalid_g (20103)`

g contains an invalid floating point value, i.e. a `NaN` or an infinite value.

`rescode.err_invalid_b (20150)`

b contains an invalid floating point value, i.e. a `NaN` or an infinite value.

`rescode.err_domain_invalid_index (20400)`

A domain index is invalid.

`rescode.err_domain_dimension (20401)`

A domain dimension is invalid.

`rescode.err_domain_dimension_psd (20402)`

A PSD domain dimension is invalid.

`rescode.err_not_power_domain (20403)`

The function is only applicable to primal and dual power cone domains.

`rescode.err_domain_power_invalid_alpha (20404)`

Alpha contains an invalid floating point value, i.e. a `NaN` or an infinite value.

`rescode.err_domain_power_negative_alpha (20405)`

Alpha contains a negative value or zero.

`rescode.err_domain_power_nleft (20406)`

The value of n_{left} is not in $[1, n-1]$ where n is the dimension.

`rescode.err_afe_invalid_index (20500)`

An affine expression index is invalid.

`rescode.err_acc_invalid_index (20600)`

A affine conic constraint index is invalid.

`rescode.err_acc_invalid_entry_index (20601)`

The index of an element in an affine conic constraint is invalid.

`rescode.err_acc_afe_domain_mismatch (20602)`

There is a mismatch between between the number of affine expressions and total dimension of the domain(s).

`rescode.err_djc_invalid_index (20700)`

A disjunctive constraint index is invalid.

`rescode.err_djc_unsupported_domain_type (20701)`

An unsupported domain type has been used in a disjunctive constraint.

`rescode.err_djc_afe_domain_mismatch (20702)`

There is a mismatch between the number of affine expressions and total dimension of the domain(s).

`rescode.err_djc_invalid_term_size (20703)`

A termize is invalid.

`rescode.err_djc_domain_termsize_mismatch (20704)`

There is a mismatch between the number of domains and the term sizes.

`rescode.err_djc_total_num_terms_mismatch (20705)`

There total number of terms in all domains does not match.

`rescode.err_undef_solution (22000)`

MOSEK has the following solution types:

- an interior-point solution,
- a basic solution,
- and an integer solution.

Each optimizer may set one or more of these solutions; e.g by default a successful optimization with the interior-point optimizer defines the interior-point solution and, for linear problems, also the basic solution. This error occurs when asking for a solution or for information about a solution that is not defined.

`rescode.err_no_doty (22010)`

No doty is available

15.9 Enumerations

`basindtype`

Basis identification

`basindtype.never`

Never do basis identification.

`basindtype.always`

Basis identification is always performed even if the interior-point optimizer terminates abnormally.

`basindtype.no_error`

Basis identification is performed if the interior-point optimizer terminates without an error.

`basindtype.if_feasible`

Basis identification is not performed if the interior-point optimizer terminates with a problem status saying that the problem is primal or dual infeasible.

`basindtype.reserved`

Not currently in use.

`boundkey`

Bound keys

`boundkey.lo`

The constraint or variable has a finite lower bound and an infinite upper bound.

`boundkey.up`

The constraint or variable has an infinite lower bound and an finite upper bound.

`boundkey.fx`

The constraint or variable is fixed.

`boundkey.fr`

The constraint or variable is free.

`boundkey.ra`

The constraint or variable is ranged.

`mark`

Mark

`mark.lo`

The lower bound is selected for sensitivity analysis.

`mark.up`

The upper bound is selected for sensitivity analysis.

`simprecision`

Experimental. Usage not recommended.

`simprecision.normal`

Experimental. Usage not recommended.

`simprecision.extended`

Experimental. Usage not recommended.

`simdegen`

Degeneracy strategies

`simdegen.none`

The simplex optimizer should use no degeneration strategy.

`simdegen.free`

The simplex optimizer chooses the degeneration strategy.

`simdegen.aggressive`

The simplex optimizer should use an aggressive degeneration strategy.

`simdegen.moderate`

The simplex optimizer should use a moderate degeneration strategy.

`simdegen.minimum`

The simplex optimizer should use a minimum degeneration strategy.

`transpose`

Transposed matrix.

`transpose.no`

No transpose is applied.

`transpose.yes`

A transpose is applied.

`uplo`

Triangular part of a symmetric matrix.

`uplo.lo`

Lower part.

`uplo.up`

Upper part.

`simreform`

Problem reformulation.

`simreform.on`

Allow the simplex optimizer to reformulate the problem.

`simreform.off`

Disallow the simplex optimizer to reformulate the problem.

`simreform.free`

The simplex optimizer can choose freely.

`simreform.aggressive`

The simplex optimizer should use an aggressive reformulation strategy.

`simdupvec`

Exploit duplicate columns.

`simdupvec.on`

Allow the simplex optimizer to exploit duplicated columns.

`simdupvec.off`

Disallow the simplex optimizer to exploit duplicated columns.

`simdupvec.free`

The simplex optimizer can choose freely.

`simhotstart`

Hot-start type employed by the simplex optimizer

`simhotstart.none`

The simplex optimizer performs a coldstart.

`simhotstart.free`

The simplex optimize chooses the hot-start type.

`simhotstart.status_keys`

Only the status keys of the constraints and variables are used to choose the type of hot-start.

`intpnthotstart`

Hot-start type employed by the interior-point optimizers.

`intpnthotstart.none`

The interior-point optimizer performs a coldstart.

`intpnthotstart.primal`

The interior-point optimizer exploits the primal solution only.

`intpnthotstart.dual`

The interior-point optimizer exploits the dual solution only.

`intpnthotstart.primal_dual`

The interior-point optimizer exploits both the primal and dual solution.

`callbackcode`

Progress callback codes

`callbackcode.begin_bi`

The basis identification procedure has been started.

`callbackcode.begin_conic`

The callback function is called when the conic optimizer is started.

`callbackcode.begin_dual_bi`

The callback function is called from within the basis identification procedure when the dual phase is started.

`callbackcode.begin_dual_sensitivity`

Dual sensitivity analysis is started.

`callbackcode.begin_dual_setup_bi`

The callback function is called when the dual BI phase is started.

`callbackcode.begin_dual_simplex`

The callback function is called when the dual simplex optimizer started.

`callbackcode.begin_dual_simplex_bi`

The callback function is called from within the basis identification procedure when the dual simplex clean-up phase is started.

`callbackcode.begin_folding`

The calback function is called at the beginning of folding.

`callbackcode.begin_folding_bi`

TBD

`callbackcode.begin_folding_bi_dual`

TBD

`callbackcode.begin_folding_bi_initialize`

TBD

`callbackcode.begin_folding_bi_optimizer`

TBD

`callbackcode.begin_folding_bi_primal`

TBD

`callbackcode.begin_infeas_ana`

The callback function is called when the infeasibility analyzer is started.

`callbackcode.begin_initialize_bi`

The callback function is called from within the basis identification procedure when the initialization phase is started.

`callbackcode.begin_intpnt`

The callback function is called when the interior-point optimizer is started.

`callbackcode.begin_license_wait`

Begin waiting for license.

`callbackcode.begin_mio`

The callback function is called when the mixed-integer optimizer is started.

`callbackcode.begin_optimize_bi`

TBD.

`callbackcode.begin_optimizer`

The callback function is called when the optimizer is started.

`callbackcode.begin_presolve`

The callback function is called when the presolve is started.

`callbackcode.begin_primal_bi`

The callback function is called from within the basis identification procedure when the primal phase is started.

`callbackcode.begin_primal_repair`

Begin primal feasibility repair.

`callbackcode.begin_primal_sensitivity`

Primal sensitivity analysis is started.

`callbackcode.begin_primal_setup_bi`

The callback function is called when the primal BI setup is started.

`callbackcode.begin_primal_simplex`

The callback function is called when the primal simplex optimizer is started.

`callbackcode.begin_primal_simplex_bi`

The callback function is called from within the basis identification procedure when the primal simplex clean-up phase is started.

`callbackcode.begin_qcqo_reformulate`

Begin QCQO reformulation.

`callbackcode.begin_read`

MOSEK has started reading a problem file.

`callbackcode.begin_root_cutgen`

The callback function is called when root cut generation is started.

`callbackcode.begin_simplex`

The callback function is called when the simplex optimizer is started.

`callbackcode.begin_solve_root_relax`

The callback function is called when solution of root relaxation is started.

`callbackcode.begin_to_conic`

Begin conic reformulation.

`callbackcode.begin_write`

MOSEK has started writing a problem file.

`callbackcode.conic`

The callback function is called from within the conic optimizer after the information database has been updated.

`callbackcode.decomp_mio`

The callback function is called when the dedicated algorithm for independent blocks inside the mixed-integer solver is started.

`callbackcode.dual_simplex`

The callback function is called from within the dual simplex optimizer.

`callbackcode.end_bi`

The callback function is called when the basis identification procedure is terminated.

`callbackcode.end_conic`

The callback function is called when the conic optimizer is terminated.

`callbackcode.end_dual_bi`

The callback function is called from within the basis identification procedure when the dual phase is terminated.

`callbackcode.end_dual_sensitivity`

Dual sensitivity analysis is terminated.

`callbackcode.end_dual_setup_bi`

The callback function is called when the dual BI phase is terminated.

`callbackcode.end_dual_simplex`

The callback function is called when the dual simplex optimizer is terminated.

`callbackcode.end_dual_simplex_bi`

The callback function is called from within the basis identification procedure when the dual clean-up phase is terminated.

`callbackcode.end_folding`

The calback function is called at the end of folding.

`callbackcode.end_folding_bi`

TBD

`callbackcode.end_folding_bi_dual`

TBD

`callbackcode.end_folding_bi_initialize`

TBD

`callbackcode.end_folding_bi_optimizer`

TBD

`callbackcode.end_folding_bi_primal`

TBD

`callbackcode.end_infeas_ana`

The callback function is called when the infeasibility analyzer is terminated.

`callbackcode.end_initialize_bi`

The callback function is called from within the basis identification procedure when the initialization phase is terminated.

`callbackcode.end_intpnt`

The callback function is called when the interior-point optimizer is terminated.

`callbackcode.end_license_wait`

End waiting for license.

`callbackcode.end_mio`

The callback function is called when the mixed-integer optimizer is terminated.

`callbackcode.end_optimize_bi`

TBD.

`callbackcode.end_optimizer`

The callback function is called when the optimizer is terminated.

`callbackcode.end_presolve`

The callback function is called when the presolve is completed.

`callbackcode.end_primal_bi`

The callback function is called from within the basis identification procedure when the primal phase is terminated.

`callbackcode.end_primal_repair`

End primal feasibility repair.

`callbackcode.end_primal_sensitivity`

Primal sensitivity analysis is terminated.

`callbackcode.end_primal_setup_bi`

The callback function is called when the primal BI setup is terminated.

`callbackcode.end_primal_simplex`

The callback function is called when the primal simplex optimizer is terminated.

`callbackcode.end_primal_simplex_bi`

The callback function is called from within the basis identification procedure when the primal clean-up phase is terminated.

`callbackcode.end_qcqo_reformulate`

End QCQO reformulation.

`callbackcode.end_read`

MOSEK has finished reading a problem file.

`callbackcode.end_root_cutgen`

The callback function is called when root cut generation is terminated.

`callbackcode.end_simplex`

The callback function is called when the simplex optimizer is terminated.

`callbackcode.end_simplex_bi`

The callback function is called from within the basis identification procedure when the simplex clean-up phase is terminated.

`callbackcode.end_solve_root_relax`

The callback function is called when solution of root relaxation is terminated.

`callbackcode.end_to_conic`

End conic reformulation.

`callbackcode.end_write`

MOSEK has finished writing a problem file.

`callbackcode.folding_bi_dual`

TBD

`callbackcode.folding_bi_optimizer`

TBD

`callbackcode.folding_bi_primal`

TBD

`callbackcode.heartbeat`

A heartbeat callback.

`callbackcode.im_dual_sensivity`

The callback function is called at an intermediate stage of the dual sensitivity analysis.

`callbackcode.im_dual_simplex`

The callback function is called at an intermediate point in the dual simplex optimizer.

`callbackcode.im_license_wait`

MOSEK is waiting for a license.

`callbackcode.im_lu`

The callback function is called from within the LU factorization procedure at an intermediate point.

`callbackcode.im_mio`

The callback function is called at an intermediate point in the mixed-integer optimizer.

`callbackcode.im_mio_dual_simplex`

The callback function is called at an intermediate point in the mixed-integer optimizer while running the dual simplex optimizer.

`callbackcode.im_mio_intpnt`

The callback function is called at an intermediate point in the mixed-integer optimizer while running the interior-point optimizer.

`callbackcode.im_mio_primal_simplex`

The callback function is called at an intermediate point in the mixed-integer optimizer while running the primal simplex optimizer.

`callbackcode.im_order`

The callback function is called from within the matrix ordering procedure at an intermediate point.

`callbackcode.im_primal_sensivity`

The callback function is called at an intermediate stage of the primal sensitivity analysis.

`callbackcode.im_primal_simplex`

The callback function is called at an intermediate point in the primal simplex optimizer.

`callbackcode.im_read`

Intermediate stage in reading.

`callbackcode.im_root_cutgen`

The callback is called from within root cut generation at an intermediate stage.

`callbackcode.im_simplex`

The callback function is called from within the simplex optimizer at an intermediate point.

`callbackcode.intpnt`

The callback function is called from within the interior-point optimizer after the information database has been updated.

`callbackcode.new_int_mio`

The callback function is called after a new integer solution has been located by the mixed-integer optimizer.

`callbackcode.optimize_bi`

TBD.

`callbackcode.primal_simplex`

The callback function is called from within the primal simplex optimizer.

`callbackcode.qo_reformulate`

The callback function is called at an intermediate stage of the conic quadratic reformulation.

`callbackcode.read_opf`

The callback function is called from the OPF reader.

`callbackcode.read_opf_section`

A chunk of Q non-zeros has been read from a problem file.

`callbackcode.restart_mio`

The callback function is called when the mixed-integer optimizer is restarted.

`callbackcode.solving_remote`

The callback function is called while the task is being solved on a remote server.

`callbackcode.update_dual_bi`

The callback function is called from within the basis identification procedure at an intermediate point in the dual phase.

`callbackcode.update_dual_simplex`

The callback function is called in the dual simplex optimizer.

`callbackcode.update_dual_simplex_bi`

The callback function is called from within the basis identification procedure at an intermediate point in the dual simplex clean-up phase. The frequency of the callbacks is controlled by the *iparam.log_sim_freq* parameter.

`callbackcode.update_presolve`

The callback function is called from within the presolve procedure.

`callbackcode.update_primal_bi`

The callback function is called from within the basis identification procedure at an intermediate point in the primal phase.

`callbackcode.update_primal_simplex`

The callback function is called in the primal simplex optimizer.

`callbackcode.update_primal_simplex_bi`

The callback function is called from within the basis identification procedure at an intermediate point in the primal simplex clean-up phase. The frequency of the callbacks is controlled by the *iparam.log_sim_freq* parameter.

`callbackcode.update_simplex`

The callback function is called from simplex optimizer.

`callbackcode.write_opf`

The callback function is called from the OPF writer.

`compresstype`

Compression types

`compresstype.none`

No compression is used.

`compresstype.free`

The type of compression used is chosen automatically.

`compresstype.gzip`

The type of compression used is gzip compatible.

`compresstype.zstd`

The type of compression used is zstd compatible.

`conetype`

Cone types

`conetype.quad`

The cone is a quadratic cone.

`conetype.rquad`

The cone is a rotated quadratic cone.

`conetype.pexp`

A primal exponential cone.

`conetype.dexp`

A dual exponential cone.

`conetype.ppow`

A primal power cone.

`conetype.dpow`

A dual power cone.

`conetype.zero`

The zero cone.

`domaintype`

Cone types

`domaintype.r`

R.

`domaintype.rzero`

The zero vector.

`domaintype.rplus`

The positive orthant.

`domaintype.rminus`

The negative orthant.

`domaintype.quadratic_cone`

The quadratic cone.

`domaintype.rquadratic_cone`
: The rotated quadratic cone.

`domaintype.primal_exp_cone`
: The primal exponential cone.

`domaintype.dual_exp_cone`
: The dual exponential cone.

`domaintype.primal_power_cone`
: The primal power cone.

`domaintype.dual_power_cone`
: The dual power cone.

`domaintype.primal_geo_mean_cone`
: The primal geometric mean cone.

`domaintype.dual_geo_mean_cone`
: The dual geometric mean cone.

`domaintype.svec_psd_cone`
: The vectorized positive semidefinite cone.

`nametype`
: Name types

`nametype.gen`
: General names. However, no duplicate and blank names are allowed.

`nametype.mps`
: MPS type names.

`nametype.lp`
: LP type names.

`symmattype`
: Cone types

`symmattype.sparse`
: Sparse symmetric matrix.

`dataformat`
: Data format types

`dataformat.extension`
: The file extension is used to determine the data file format.

`dataformat.mps`
: The data file is MPS formatted.

`dataformat.lp`
: The data file is LP formatted.

`dataformat.op`
: The data file is an optimization problem formatted file.

`dataformat.free_mps`
: The data a free MPS formatted file.

`dataformat.task`
: Generic task dump file.

`dataformat.ptf`
: (P)retty (T)ext (F)format.

`dataformat.cb`
: Conic benchmark format,

`dataformat.json_task`

JSON based task format.

`solformat`

Data format types

`solformat.extension`

The file extension is used to determine the data file format.

`solformat.b`

Simple binary format

`solformat.task`

Tar based format.

`solformat.json_task`

JSON based format.

`dinfitem`

Double information items

`dinfitem.ana_pro_scalarized_constraint_matrix_density`

Density percentage of the scalarized constraint matrix.

`dinfitem.bi_clean_time`

Time spent within the clean-up phase of the basis identification procedure since its invocation (in seconds).

`dinfitem.bi_dual_time`

Time spent within the dual phase basis identification procedure since its invocation (in seconds).

`dinfitem.bi_primal_time`

Time spent within the primal phase of the basis identification procedure since its invocation (in seconds).

`dinfitem.bi_time`

Time spent within the basis identification procedure since its invocation (in seconds).

`dinfitem.folding_bi_optimize_time`

TBD

`dinfitem.folding_bi_unfold_dual_time`

TBD

`dinfitem.folding_bi_unfold_initialize_time`

TBD

`dinfitem.folding_bi_unfold_primal_time`

TBD

`dinfitem.folding_bi_unfold_time`

TBD

`dinfitem.folding_factor`

Problem size after folding as a fraction of the original size.

`dinfitem.folding_time`

Total time spent in folding for continuous problems (in seconds).

`dinfitem.intpnt_dual_feas`

Dual feasibility measure reported by the interior-point optimizer. (For the interior-point optimizer this measure is not directly related to the original problem because a homogeneous model is employed.)

`dinfitem.intpnt_dual_obj`

Dual objective value reported by the interior-point optimizer.

`dinfitem.intpnt_factor_num_flops`

An estimate of the number of flops used in the factorization.

`dinfitem.intpnt_opt_status`

A measure of optimality of the solution. It should converge to $+1$ if the problem has a primal-dual optimal solution, and converge to -1 if the problem is (strictly) primal or dual infeasible. If the measure converges to another constant, or fails to settle, the problem is usually ill-posed.

`dinfitem.intpnt_order_time`

Order time (in seconds).

`dinfitem.intpnt_primal_feas`

Primal feasibility measure reported by the interior-point optimizer. (For the interior-point optimizer this measure is not directly related to the original problem because a homogeneous model is employed).

`dinfitem.intpnt_primal_obj`

Primal objective value reported by the interior-point optimizer.

`dinfitem.intpnt_time`

Time spent within the interior-point optimizer since its invocation (in seconds).

`dinfitem.mio_clique_selection_time`

Selection time for clique cuts (in seconds).

`dinfitem.mio_clique_separation_time`

Separation time for clique cuts (in seconds).

`dinfitem.mio_cmir_selection_time`

Selection time for CMIR cuts (in seconds).

`dinfitem.mio_cmir_separation_time`

Separation time for CMIR cuts (in seconds).

`dinfitem.mio_construct_solution_obj`

If **MOSEK** has successfully constructed an integer feasible solution, then this item contains the optimal objective value corresponding to the feasible solution.

`dinfitem.mio_dual_bound_after_presolve`

Value of the dual bound after presolve but before cut generation.

`dinfitem.mio_gmi_selection_time`

Selection time for GMI cuts (in seconds).

`dinfitem.mio_gmi_separation_time`

Separation time for GMI cuts (in seconds).

`dinfitem.mio_implied_bound_selection_time`

Selection time for implied bound cuts (in seconds).

`dinfitem.mio_implied_bound_separation_time`

Separation time for implied bound cuts (in seconds).

`dinfitem.mio_initial_feasible_solution_obj`

If the user provided solution was found to be feasible this information item contains it's objective value.

`dinfitem.mio_knapsack_cover_selection_time`

Selection time for knapsack cover (in seconds).

`dinfitem.mio_knapsack_cover_separation_time`

Separation time for knapsack cover (in seconds).

`dinfitem.mio_lipro_selection_time`

Selection time for lift-and-project cuts (in seconds).

`dinfitem.mio_lipro_separation_time`

Separation time for lift-and-project cuts (in seconds).

`dinfitem.mio_obj_abs_gap`

Given the mixed-integer optimizer has computed a feasible solution and a bound on the optimal objective value, then this item contains the absolute gap defined by

$$|(\text{objective value of feasible solution}) - (\text{objective bound})|.$$

Otherwise it has the value -1.0.

`dinfitem.mio_obj_bound`

The best known bound on the objective function. This value is undefined until at least one relaxation has been solved: To see if this is the case check that *iinfitem.mio_num_relax* is strictly positive.

`dinfitem.mio_obj_int`

The primal objective value corresponding to the best integer feasible solution. Please note that at least one integer feasible solution must have been located i.e. check *iinfitem.mio_num_int_solutions*.

`dinfitem.mio_obj_rel_gap`

Given that the mixed-integer optimizer has computed a feasible solution and a bound on the optimal objective value, then this item contains the relative gap defined by

$$\frac{|(\text{objective value of feasible solution}) - (\text{objective bound})|}{\max(\delta, |(\text{objective value of feasible solution})|)}.$$

where δ is given by the parameter *dparam.mio_rel_gap_const*. Otherwise it has the value -1.0.

`dinfitem.mio_probing_time`

Total time for probing (in seconds).

`dinfitem.mio_root_cut_selection_time`

Total time for cut selection (in seconds).

`dinfitem.mio_root_cut_separation_time`

Total time for cut separation (in seconds).

`dinfitem.mio_root_optimizer_time`

Time spent in the contiuous optimizer while processing the root node relaxation (in seconds).

`dinfitem.mio_root_presolve_time`

Time spent presolving the problem at the root node (in seconds).

`dinfitem.mio_root_time`

Time spent processing the root node (in seconds).

`dinfitem.mio_symmetry_detection_time`

Total time for symmetry detection (in seconds).

`dinfitem.mio_symmetry_factor`

Degree to which the problem is affected by detected symmetry.

`dinfitem.mio_time`

Time spent in the mixed-integer optimizer (in seconds).

`dinfitem.mio_user_obj_cut`

If the objective cut is used, then this information item has the value of the cut.

`dinfitem.optimizer_ticks`

Total number of ticks spent in the optimizer since it was invoked. It is strictly negative if it is not available.

`dinfitem.optimizer_time`

Total time spent in the optimizer since it was invoked (in seconds).

`dinfitem.presolve_eli_time`

Total time spent in the eliminator since the presolve was invoked (in seconds).

`dinfitem.presolve_lindep_time`

Total time spent in the linear dependency checker since the presolve was invoked (in seconds).

`dinfitem.presolve_time`

Total time spent in the presolve since it was invoked (in seconds).

`dinfitem.presolve_total_primal_perturbation`

Total perturbation of the bounds of the primal problem.

`dinfitem.primal_repair_penalty_obj`

The optimal objective value of the penalty function.

`dinfitem.qcqo_reformulate_max_perturbation`

Maximum absolute diagonal perturbation occurring during the QCQO reformulation.

`dinfitem.qcqo_reformulate_time`

Time spent with conic quadratic reformulation (in seconds).

`dinfitem.qcqo_reformulate_worst_cholesky_column_scaling`

Worst Cholesky column scaling.

`dinfitem.qcqo_reformulate_worst_cholesky_diag_scaling`

Worst Cholesky diagonal scaling.

`dinfitem.read_data_time`

Time spent reading the data file (in seconds).

`dinfitem.remote_time`

The total real time in seconds spent when optimizing on a server by the process performing the optimization on the server (in seconds).

`dinfitem.sim_dual_time`

Time spent in the dual simplex optimizer since invoking it (in seconds).

`dinfitem.sim_feas`

Feasibility measure reported by the simplex optimizer.

`dinfitem.sim_obj`

Objective value reported by the simplex optimizer.

`dinfitem.sim_primal_time`

Time spent in the primal simplex optimizer since invoking it (in seconds).

`dinfitem.sim_time`

Time spent in the simplex optimizer since invoking it (in seconds).

`dinfitem.sol_bas_dual_obj`

Dual objective value of the basic solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.updatesolutioninfo*.

`dinfitem.sol_bas_dviolcon`

Maximal dual bound violation for x^c in the basic solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.updatesolutioninfo*.

`dinfitem.sol_bas_dviolvar`

Maximal dual bound violation for x^x in the basic solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.updatesolutioninfo*.

`dinfitem.sol_bas_nrm_barx`

Infinity norm of $\overline{X}$ in the basic solution.

`dinfitem.sol_bas_nrm_slc`

Infinity norm of s_l^c in the basic solution.

`dinfitem.sol_bas_nrm_slx`

Infinity norm of s_l^x in the basic solution.

`dinfitem.sol_bas_nrm_suc`

Infinity norm of s_u^c in the basic solution.

`dinfitem.sol_bas_nrm_sux`

Infinity norm of s_u^X in the basic solution.

`dinfitem.sol_bas_nrm_xc`

Infinity norm of x^c in the basic solution.

`dinfitem.sol_bas_nrm_xx`

Infinity norm of x^x in the basic solution.

`dinfitem.sol_bas_nrm_y`

Infinity norm of y in the basic solution.

`dinfitem.sol_bas_primal_obj`

Primal objective value of the basic solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.updatesolutioninfo*.

`dinfitem.sol_bas_pviolcon`

Maximal primal bound violation for x^c in the basic solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.updatesolutioninfo*.

`dinfitem.sol_bas_pviolvar`

Maximal primal bound violation for x^x in the basic solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.updatesolutioninfo*.

`dinfitem.sol_itg_nrm_barx`

Infinity norm of $\overline{X}$ in the integer solution.

`dinfitem.sol_itg_nrm_xc`

Infinity norm of x^c in the integer solution.

`dinfitem.sol_itg_nrm_xx`

Infinity norm of x^x in the integer solution.

`dinfitem.sol_itg_primal_obj`

Primal objective value of the integer solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.updatesolutioninfo*.

`dinfitem.sol_itg_pviolacc`

Maximal primal violation for affine conic constraints in the integer solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.updatesolutioninfo*.

`dinfitem.sol_itg_pviolbarvar`

Maximal primal bound violation for $\overline{X}$ in the integer solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.updatesolutioninfo*.

`dinfitem.sol_itg_pviolcon`

Maximal primal bound violation for x^c in the integer solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.updatesolutioninfo*.

`dinfitem.sol_itg_pviolcones`

Maximal primal violation for primal conic constraints in the integer solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.updatesolutioninfo*.

`dinfitem.sol_itg_pvioldjc`

Maximal primal violation for disjunctive constraints in the integer solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.updatesolutioninfo*.

`dinfitem.sol_itg_pviolitg`

Maximal violation for the integer constraints in the integer solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.updatesolutioninfo*.

`dinfitem.sol_itg_pviolvar`

Maximal primal bound violation for x^x in the integer solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.updatesolutioninfo*.

`dinfitem.sol_itr_dual_obj`

Dual objective value of the interior-point solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.updatesolutioninfo*.

`dinfitem.sol_itr_dviolacc`

Maximal dual violation for the affine conic constraints in the interior-point solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.updatesolutioninfo*.

`dinfitem.sol_itr_dviolbarvar`

Maximal dual bound violation for $\overline{X}$ in the interior-point solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.updatesolutioninfo*.

`dinfitem.sol_itr_dviolcon`

Maximal dual bound violation for x^c in the interior-point solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.updatesolutioninfo*.

`dinfitem.sol_itr_dviolcones`

Maximal dual violation for conic constraints in the interior-point solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.updatesolutioninfo*.

`dinfitem.sol_itr_dviolvar`

Maximal dual bound violation for x^x in the interior-point solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.updatesolutioninfo*.

`dinfitem.sol_itr_nrm_bars`

Infinity norm of $\overline{S}$ in the interior-point solution.

`dinfitem.sol_itr_nrm_barx`

Infinity norm of $\overline{X}$ in the interior-point solution.

`dinfitem.sol_itr_nrm_slc`

Infinity norm of s_l^c in the interior-point solution.

`dinfitem.sol_itr_nrm_slx`

Infinity norm of s_l^x in the interior-point solution.

`dinfitem.sol_itr_nrm_snx`

Infinity norm of s_n^x in the interior-point solution.

`dinfitem.sol_itr_nrm_suc`

Infinity norm of s_u^c in the interior-point solution.

`dinfitem.sol_itr_nrm_sux`

Infinity norm of s_u^X in the interior-point solution.

`dinfitem.sol_itr_nrm_xc`

Infinity norm of x^c in the interior-point solution.

`dinfitem.sol_itr_nrm_xx`

Infinity norm of x^x in the interior-point solution.

`dinfitem.sol_itr_nrm_y`

Infinity norm of y in the interior-point solution.

`dinfitem.sol_itr_primal_obj`

Primal objective value of the interior-point solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.updatesolutioninfo*.

`dinfitem.sol_itr_pviolacc`

Maximal primal violation for affine conic constraints in the interior-point solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.updatesolutioninfo*.

`dinfitem.sol_itr_pviolbarvar`

Maximal primal bound violation for $\overline{X}$ in the interior-point solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.updatesolutioninfo*.

`dinfitem.sol_itr_pviolcon`

Maximal primal bound violation for x^c in the interior-point solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.updatesolutioninfo*.

`dinfitem.sol_itr_pviolcones`

Maximal primal violation for conic constraints in the interior-point solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.updatesolutioninfo*.

`dinfitem.sol_itr_pviolvar`

Maximal primal bound violation for x^x in the interior-point solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.updatesolutioninfo*.

`dinfitem.to_conic_time`

Time spent in the last to conic reformulation (in seconds).

`dinfitem.write_data_time`

Time spent writing the data file (in seconds).

`feature`

License feature

`feature.pts`

Base system.

`feature.pton`

Conic extension.

`liinfitem`

Long integer information items.

`liinfitem.ana_pro_scalarized_constraint_matrix_num_columns`

Number of columns in the scalarized constraint matrix.

`liinfitem.ana_pro_scalarized_constraint_matrix_num_nz`

Number of non-zero entries in the scalarized constraint matrix.

`liinfitem.ana_pro_scalarized_constraint_matrix_num_rows`

Number of rows in the scalarized constraint matrix.

`liinfitem.bi_clean_iter`

Number of clean iterations performed in the basis identification.

`liinfitem.bi_dual_iter`

Number of dual pivots performed in the basis identification.

`liinfitem.bi_primal_iter`

Number of primal pivots performed in the basis identification.

`liinfitem.folding_bi_dual_iter`

TBD

`liinfitem.folding_bi_optimizer_iter`

TBD

`liinfitem.folding_bi_primal_iter`

TBD

`liinfitem.intpnt_factor_num_nz`

Number of non-zeros in factorization.

`liinfitem.mio_anz`

Number of non-zero entries in the constraint matrix of the problem to be solved by the mixed-integer optimizer.

`liinfitem.mio_final_anz`

Number of non-zero entries in the constraint matrix of the mixed-integer optimizer's final problem.

`liinfitem.mio_intpnt_iter`

Number of interior-point iterations performed by the mixed-integer optimizer.

`liinfitem.mio_num_dual_illposed_cer`

Number of dual illposed certificates encountered by the mixed-integer optimizer.

`liinfitem.mio_num_prim_illposed_cer`

Number of primal illposed certificates encountered by the mixed-integer optimizer.

`liinfitem.mio_presolved_anz`

Number of non-zero entries in the constraint matrix of the problem after the mixed-integer optimizer's presolve.

`liinfitem.mio_simplex_iter`

Number of simplex iterations performed by the mixed-integer optimizer.

`liinfitem.rd_numacc`

Number of affince conic constraints.

`liinfitem.rd_numanz`

Number of non-zeros in A that is read.

`liinfitem.rd_numdjc`

Number of disjuncive constraints.

`liinfitem.rd_numqnz`

Number of Q non-zeros.

`liinfitem.simplex_iter`

Number of iterations performed by the simplex optimizer.

`iinfitem`

Integer information items.

`iinfitem.ana_pro_num_con`

Number of constraints in the problem. This value is set by *Task.analyzeproblem*.

`iinfitem.ana_pro_num_con_eq`

Number of equality constraints. This value is set by *Task.analyzeproblem*.

`iinfitem.ana_pro_num_con_fr`

Number of unbounded constraints. This value is set by *Task.analyzeproblem*.

`iinfitem.ana_pro_num_con_lo`

Number of constraints with a lower bound and an infinite upper bound. This value is set by *Task.analyzeproblem*.

`iinfitem.ana_pro_num_con_ra`

Number of constraints with finite lower and upper bounds. This value is set by *Task.analyzeproblem*.

`iinfitem.ana_pro_num_con_up`

Number of constraints with an upper bound and an infinite lower bound. This value is set by *Task.analyzeproblem*.

`iinfitem.ana_pro_num_var`

Number of variables in the problem. This value is set by *Task.analyzeproblem*.

`iinfitem.ana_pro_num_var_bin`

Number of binary (0-1) variables. This value is set by *Task.analyzeproblem*.

`iinfitem.ana_pro_num_var_cont`

Number of continuous variables. This value is set by *Task.analyzeproblem*.

`iinfitem.ana_pro_num_var_eq`

Number of fixed variables. This value is set by *Task.analyzeproblem*.

`iinfitem.ana_pro_num_var_fr`

Number of free variables. This value is set by *Task.analyzeproblem*.

`iinfitem.ana_pro_num_var_int`

Number of general integer variables. This value is set by *Task.analyzeproblem*.

`iinfitem.ana_pro_num_var_lo`

Number of variables with a lower bound and an infinite upper bound. This value is set by *Task.analyzeproblem*.

`iinfitem.ana_pro_num_var_ra`

Number of variables with finite lower and upper bounds. This value is set by *Task.analyzeproblem*.

`iinfitem.ana_pro_num_var_up`

Number of variables with an upper bound and an infinite lower bound. This value is set by *Task.analyzeproblem*.

`iinfitem.folding_applied`

Non-zero if folding was exploited.

`iinfitem.intpnt_factor_dim_dense`

Dimension of the dense sub system in factorization.

`iinfitem.intpnt_iter`

Number of interior-point iterations since invoking the interior-point optimizer.

`iinfitem.intpnt_num_threads`

Number of threads that the interior-point optimizer is using.

`iinfitem.intpnt_solve_dual`

Non-zero if the interior-point optimizer is solving the dual problem.

`iinfitem.mio_absgap_satisfied`

Non-zero if absolute gap is within tolerances.

`iinfitem.mio_clique_table_size`

Size of the clique table.

`iinfitem.mio_construct_solution`

This item informs if **MOSEK** constructed an initial integer feasible solution.

- -1: tried, but failed,
- 0: no partial solution supplied by the user,
- 1: constructed feasible solution.

`iinfitem.mio_final_numbin`

Number of binary variables in the mixed-integer optimizer's final problem.

`iinfitem.mio_final_numbinconevar`

Number of binary cone variables in the mixed-integer optimizer's final problem.

`iinfitem.mio_final_numcon`

Number of constraints in the mixed-integer optimizer's final problem.

`iinfitem.mio_final_numcone`

Number of cones in the mixed-integer optimizer's final problem.

`iinfitem.mio_final_numconevar`

Number of cone variables in the mixed-integer optimizer's final problem.

`iinfitem.mio_final_numcont`

Number of continuous variables in the mixed-integer optimizer's final problem.

`iinfitem.mio_final_numcontconevar`

Number of continuous cone variables in the mixed-integer optimizer's final problem.

`iinfitem.mio_final_numdexpcones`

Number of dual exponential cones in the mixed-integer optimizer's final problem.

`iinfitem.mio_final_numdjc`

Number of disjunctive constraints in the mixed-integer optimizer's final problem.

`iinfitem.mio_final_numdpowcones`

Number of dual power cones in the mixed-integer optimizer's final problem.

`iinfitem.mio_final_numint`

Number of integer variables in the mixed-integer optimizer's final problem.

`iinfitem.mio_final_numintconevar`

Number of integer cone variables in the mixed-integer optimizer's final problem.

`iinfitem.mio_final_numpexpcones`

Number of primal exponential cones in the mixed-integer optimizer's final problem.

`iinfitem.mio_final_numppowcones`

Number of primal power cones in the mixed-integer optimizer's final problem.

`iinfitem.mio_final_numqcones`

Number of quadratic cones in the mixed-integer optimizer's final problem.

`iinfitem.mio_final_numrqcones`

Number of rotated quadratic cones in the mixed-integer optimizer's final problem.

`iinfitem.mio_final_numvar`

Number of variables in the mixed-integer optimizer's final problem.

`iinfitem.mio_initial_feasible_solution`

This item informs if **MOSEK** found the solution provided by the user to be feasible

- 0: solution provided by the user was not found to be feasible for the current problem,
- 1: user provided solution was found to be feasible.

`iinfitem.mio_node_depth`

Depth of the last node solved.

`iinfitem.mio_num_active_nodes`

Number of active branch and bound nodes.

`iinfitem.mio_num_active_root_cuts`

Number of active cuts in the final relaxation after the mixed-integer optimizer's root cut generation.

`iinfitem.mio_num_blocks_solved_in_bb`

Number of independent decomposition blocks solved though a dedicated algorithm.

`iinfitem.mio_num_blocks_solved_in_presolve`

Number of independent decomposition blocks solved during presolve.

`iinfitem.mio_num_branch`

Number of branches performed during the optimization.

`iinfitem.mio_num_int_solutions`

Number of integer feasible solutions that have been found.

`iinfitem.mio_num_relax`

Number of relaxations solved during the optimization.

`iinfitem.mio_num_repeated_presolve`

Number of times presolve was repeated at root.

`iinfitem.mio_num_restarts`

Number of restarts performed during the optimization.

`iinfitem.mio_num_root_cut_rounds`

Number of cut separation rounds at the root node of the mixed-integer optimizer.

`iinfitem.mio_num_selected_clique_cuts`

Number of clique cuts selected to be included in the relaxation.

`iinfitem.mio_num_selected_cmir_cuts`

Number of Complemented Mixed Integer Rounding (CMIR) cuts selected to be included in the relaxation.

`iinfitem.mio_num_selected_gomory_cuts`

Number of Gomory cuts selected to be included in the relaxation.

`iinfitem.mio_num_selected_implied_bound_cuts`

Number of implied bound cuts selected to be included in the relaxation.

`iinfitem.mio_num_selected_knapsack_cover_cuts`

Number of clique cuts selected to be included in the relaxation.

`iinfitem.mio_num_selected_lipro_cuts`

Number of lift-and-project cuts selected to be included in the relaxation.

`iinfitem.mio_num_separated_clique_cuts`

Number of separated clique cuts.

`iinfitem.mio_num_separated_cmir_cuts`

Number of separated Complemented Mixed Integer Rounding (CMIR) cuts.

`iinfitem.mio_num_separated_gomory_cuts`

Number of separated Gomory cuts.

`iinfitem.mio_num_separated_implied_bound_cuts`

Number of separated implied bound cuts.

`iinfitem.mio_num_separated_knapsack_cover_cuts`

Number of separated clique cuts.

`iinfitem.mio_num_separated_lipro_cuts`

Number of separated lift-and-project cuts.

`iinfitem.mio_num_solved_nodes`

Number of branch and bounds nodes solved in the main branch and bound tree.

`iinfitem.mio_numbin`

Number of binary variables in the problem to be solved by the mixed-integer optimizer.

`iinfitem.mio_numbinconevar`

Number of binary cone variables in the problem to be solved by the mixed-integer optimizer.

`iinfitem.mio_numcon`

Number of constraints in the problem to be solved by the mixed-integer optimizer.

`iinfitem.mio_numcone`

Number of cones in the problem to be solved by the mixed-integer optimizer.

`iinfitem.mio_numconevar`

Number of cone variables in the problem to be solved by the mixed-integer optimizer.

`iinfitem.mio_numcont`

Number of continuous variables in the problem to be solved by the mixed-integer optimizer.

`iinfitem.mio_numcontconevar`

Number of continuous cone variables in the problem to be solved by the mixed-integer optimizer.

`iinfitem.mio_numdexpcones`

Number of dual exponential cones in the problem to be solved by the mixed-integer optimizer.

`iinfitem.mio_numdjc`

Number of disjunctive constraints in the problem to be solved by the mixed-integer optimizer.

`iinfitem.mio_numdpowcones`

Number of dual power cones in the problem to be solved by the mixed-integer optimizer.

`iinfitem.mio_numint`

Number of integer variables in the problem to be solved by the mixed-integer optimizer.

`iinfitem.mio_numintconevar`

Number of integer cone variables in the problem to be solved by the mixed-integer optimizer.

`iinfitem.mio_numpexpcones`

Number of primal exponential cones in the problem to be solved by the mixed-integer optimizer.

`iinfitem.mio_numppowcones`

Number of primal power cones in the problem to be solved by the mixed-integer optimizer.

`iinfitem.mio_numqcones`

Number of quadratic cones in the problem to be solved by the mixed-integer optimizer.

`iinfitem.mio_numrqcones`

Number of rotated quadratic cones in the problem to be solved by the mixed-integer optimizer.

`iinfitem.mio_numvar`

Number of variables in the problem to be solved by the mixed-integer optimizer.

`iinfitem.mio_obj_bound_defined`

Non-zero if a valid objective bound has been found, otherwise zero.

`iinfitem.mio_presolved_numbin`

Number of binary variables in the problem after the mixed-integer optimizer's presolve.

`iinfitem.mio_presolved_numbinconevar`

Number of binary cone variables in the problem after the mixed-integer optimizer's presolve.

`iinfitem.mio_presolved_numcon`

Number of constraints in the problem after the mixed-integer optimizer's presolve.

`iinfitem.mio_presolved_numcone`

Number of cones in the problem after the mixed-integer optimizer's presolve.

`iinfitem.mio_presolved_numconevar`

Number of cone variables in the problem after the mixed-integer optimizer's presolve.

`iinfitem.mio_presolved_numcont`

Number of continuous variables in the problem after the mixed-integer optimizer's presolve.

`iinfitem.mio_presolved_numcontconevar`

Number of continuous cone variables in the problem after the mixed-integer optimizer's presolve.

`iinfitem.mio_presolved_numdexpcones`

Number of dual exponential cones in the problem after the mixed-integer optimizer's presolve.

`iinfitem.mio_presolved_numdjc`

Number of disjunctive constraints in the problem after the mixed-integer optimizer's presolve.

`iinfitem.mio_presolved_numdpowcones`

Number of dual power cones in the problem after the mixed-integer optimizer's presolve.

`iinfitem.mio_presolved_numint`

Number of integer variables in the problem after the mixed-integer optimizer's presolve.

`iinfitem.mio_presolved_numintconevar`

Number of integer cone variables in the problem after the mixed-integer optimizer's presolve.

`iinfitem.mio_presolved_numpexpcones`

Number of primal exponential cones in the problem after the mixed-integer optimizer's presolve.

`iinfitem.mio_presolved_numppowcones`

Number of primal power cones in the problem after the mixed-integer optimizer's presolve.

`iinfitem.mio_presolved_numqcones`

Number of quadratic cones in the problem after the mixed-integer optimizer's presolve.

`iinfitem.mio_presolved_numrqcones`

Number of rotated quadratic cones in the problem after the mixed-integer optimizer's presolve.

`iinfitem.mio_presolved_numvar`

Number of variables in the problem after the mixed-integer optimizer's presolve.

`iinfitem.mio_relgap_satisfied`

Non-zero if relative gap is within tolerances.

`iinfitem.mio_total_num_selected_cuts`

Total number of cuts selected to be included in the relaxation by the mixed-integer optimizer.

`iinfitem.mio_total_num_separated_cuts`

Total number of cuts separated by the mixed-integer optimizer.

`iinfitem.mio_user_obj_cut`

If it is non-zero, then the objective cut is used.

`iinfitem.opt_numcon`

Number of constraints in the problem solved when the optimizer is called.

`iinfitem.opt_numvar`

Number of variables in the problem solved when the optimizer is called

`iinfitem.optimize_response`

The response code returned by optimize.

`iinfitem.presolve_num_primal_perturbations`

Number perturbations to thhe bounds of the primal problem.

`iinfitem.purify_dual_success`

Is nonzero if the dual solution is purified.

`iinfitem.purify_primal_success`

Is nonzero if the primal solution is purified.

`iinfitem.rd_numbarvar`

Number of symmetric variables read.

`iinfitem.rd_numcon`

Number of constraints read.

`iinfitem.rd_numcone`

Number of conic constraints read.

`iinfitem.rd_numintvar`

Number of integer-constrained variables read.

`iinfitem.rd_numq`

Number of nonempty Q matrices read.

`iinfitem.rd_numvar`

Number of variables read.

`iinfitem.rd_protype`

Problem type.

`iinfitem.sim_dual_deg_iter`

The number of dual degenerate iterations.

`iinfitem.sim_dual_hotstart`

If 1 then the dual simplex algorithm is solving from an advanced basis.

`iinfitem.sim_dual_hotstart_lu`

If 1 then a valid basis factorization of full rank was located and used by the dual simplex algorithm.

`iinfitem.sim_dual_inf_iter`

The number of iterations taken with dual infeasibility.

`iinfitem.sim_dual_iter`

Number of dual simplex iterations during the last optimization.

`iinfitem.sim_numcon`

Number of constraints in the problem solved by the simplex optimizer.

`iinfitem.sim_numvar`

Number of variables in the problem solved by the simplex optimizer.

`iinfitem.sim_primal_deg_iter`

The number of primal degenerate iterations.

`iinfitem.sim_primal_hotstart`

If 1 then the primal simplex algorithm is solving from an advanced basis.

`iinfitem.sim_primal_hotstart_lu`

If 1 then a valid basis factorization of full rank was located and used by the primal simplex algorithm.

`iinfitem.sim_primal_inf_iter`

The number of iterations taken with primal infeasibility.

`iinfitem.sim_primal_iter`

Number of primal simplex iterations during the last optimization.

`iinfitem.sim_solve_dual`

Is non-zero if dual problem is solved.

`iinfitem.sol_bas_prosta`

Problem status of the basic solution. Updated after each optimization.

`iinfitem.sol_bas_solsta`

Solution status of the basic solution. Updated after each optimization.

`iinfitem.sol_itg_prosta`

Problem status of the integer solution. Updated after each optimization.

`iinfitem.sol_itg_solsta`

Solution status of the integer solution. Updated after each optimization.

`iinfitem.sol_itr_prosta`

Problem status of the interior-point solution. Updated after each optimization.

`iinfitem.sol_itr_solsta`

Solution status of the interior-point solution. Updated after each optimization.

`iinfitem.sto_num_a_realloc`

Number of times the storage for storing A has been changed. A large value may indicates that memory fragmentation may occur.

`inftype`

Information item types

`inftype.dou_type`

Is a double information type.

`inftype.int_type`

Is an integer.

`inftype.lint_type`

Is a long integer.

`iomode`

Input/output modes

`iomode.read`

The file is read-only.

`iomode.write`

The file is write-only. If the file exists then it is truncated when it is opened. Otherwise it is created when it is opened.

`iomode.readwrite`

The file is to read and write.

`branchdir`

Specifies the branching direction.

`branchdir.free`

The mixed-integer optimizer decides which branch to choose.

`branchdir.up`

The mixed-integer optimizer always chooses the up branch first.

`branchdir.down`

The mixed-integer optimizer always chooses the down branch first.

`branchdir.near`

Branch in direction nearest to selected fractional variable.

`branchdir.far`

Branch in direction farthest from selected fractional variable.

`branchdir.root_lp`

Chose direction based on root lp value of selected variable.

`branchdir.guided`

Branch in direction of current incumbent.

`branchdir.pseudocost`

Branch based on the pseudocost of the variable.

`miqcqoreformmethod`

Specifies the reformulation method for mixed-integer quadratic problems.

`miqcqoreformmethod.free`

The mixed-integer optimizer decides which reformulation method to apply.

`miqcqoreformmethod.none`

No reformulation method is applied.

`miqcqoreformmethod.linearization`

A reformulation via linearization is applied.

`miqcqoreformmethod.eigen_val_method`

The eigenvalue method is applied.

`miqcqoreformmethod.diag_sdp`

A perturbation of matrix diagonals via the solution of SDPs is applied.

`miqcqoreformmethod.relax_sdp`

A Reformulation based on the solution of an SDP-relaxation of the problem is applied.

`miodatapermmethod`

Specifies the problem data permutation method for mixed-integer problems.

`miodatapermmethod.none`

No problem data permutation is applied.

`miodatapermmethod.cyclic_shift`

A random cyclic shift is applied to permute the problem data.

`miodatapermmethod.random`

A random permutation is applied to the problem data.

`miocontsoltype`

Continuous mixed-integer solution type

`miocontsoltype.none`

No interior-point or basic solution are reported when the mixed-integer optimizer is used.

`miocontsoltype.root`

The reported interior-point and basic solutions are a solution to the root node problem when mixed-integer optimizer is used.

`miocontsoltype.itg`

The reported interior-point and basic solutions are a solution to the problem with all integer variables fixed at the value they have in the integer solution. A solution is only reported in case the problem has a primal feasible solution.

`miocontsoltype.itg_rel`

In case the problem is primal feasible then the reported interior-point and basic solutions are a solution to the problem with all integer variables fixed at the value they have in the integer solution. If the problem is primal infeasible, then the solution to the root node problem is reported.

`miomode`

Integer restrictions

`miomode.ignored`

The integer constraints are ignored and the problem is solved as a continuous problem.

`miomode.satisfied`

Integer restrictions should be satisfied.

`mionodeseltype`

Mixed-integer node selection types

`mionodeseltype.free`

The optimizer decides the node selection strategy.

`mionodeseltype.first`

The optimizer employs a depth first node selection strategy.

`mionodeseltype.best`

The optimizer employs a best bound node selection strategy.

`mionodeseltype.pseudo`

The optimizer employs selects the node based on a pseudo cost estimate.

`miovarseltype`

Mixed-integer variable selection types

`miovarseltype.free`

The optimizer decides the variable selection strategy.

`miovarseltype.pseudocost`

The optimizer employs pseudocost variable selection.

`miovarseltype.strong`

The optimizer employs strong branching varfiable selection

`mpsformat`

MPS file format type

`mpsformat.strict`

It is assumed that the input file satisfies the MPS format strictly.

`mpsformat.relaxed`

It is assumed that the input file satisfies a slightly relaxed version of the MPS format.

`mpsformat.free`

It is assumed that the input file satisfies the free MPS format. This implies that spaces are not allowed in names. Otherwise the format is free.

`mpsformat.cplex`

The CPLEX compatible version of the MPS format is employed.

`objsense`

Objective sense types

`objsense.minimize`

The problem should be minimized.

`objsense.maximize`

The problem should be maximized.

`onoffkey`

On/off

`onoffkey.on`

Switch the option on.

`onoffkey.off`

Switch the option off.

`optimizertype`

Optimizer types

`optimizertype.conic`

The optimizer for problems having conic constraints.

`optimizertype.dual_simplex`

The dual simplex optimizer is used.

`optimizertype.free`

The optimizer is chosen automatically.

`optimizertype.free_simplex`

One of the simplex optimizers is used.

`optimizertype.intpnt`

The interior-point optimizer is used.

`optimizertype.mixed_int`

The mixed-integer optimizer.

`optimizertype.new_dual_simplex`

The new dual simplex optimizer is used.

`optimizertype.new_primal_simplex`

The new primal simplex optimizer is used. It is not recommended to use this option.

`optimizertype.primal_simplex`

The primal simplex optimizer is used.

`orderingtype`

Ordering strategies

`orderingtype.free`

The ordering method is chosen automatically.

`orderingtype.appminloc`

Approximate minimum local fill-in ordering is employed.

`orderingtype.experimental`

This option should not be used.

`orderingtype.try_graphpar`

Always try the graph partitioning based ordering.

`orderingtype.force_graphpar`

Always use the graph partitioning based ordering even if it is worse than the approximate minimum local fill ordering.

`orderingtype.none`

No ordering is used. Note using this value almost always leads to a significantly slow down.

`presolvemode`

Presolve method.

`presolvemode.off`

The problem is not presolved before it is optimized.

`presolvemode.on`

The problem is presolved before it is optimized.

`presolvemode.free`

It is decided automatically whether to presolve before the problem is optimized.

`foldingmode`

Method of folding (symmetry detection for continuous problems).

`foldingmode.off`

Disabled.

`foldingmode.free`

The solver decides on the usage and amount of folding.

`foldingmode.free_unless_basic`

If only the interior-point solution is requested then the solver decides; if the basic solution is requested then folding is disabled.

`foldingmode.force`

Full folding is always performed regardless of workload.

`parametertype`

Parameter type

`parametertype.invalid_type`

Not a valid parameter.

`parametertype.dou_type`

Is a double parameter.

`parametertype.int_type`

Is an integer parameter.

`parametertype.str_type`

Is a string parameter.

`problemitem`

Problem data items

`problemitem.var`

Item is a variable.

`problemitem.con`

Item is a constraint.

`problemitem.cone`

Item is a cone.

`problemtype`

Problem types

`problemtype.lo`

The problem is a linear optimization problem.

`problemtype.qo`

The problem is a quadratic optimization problem.

`problemtype.qcqo`

The problem is a quadratically constrained optimization problem.

`problemtype.conic`

A conic optimization.

`problemtype.mixed`

General nonlinear constraints and conic constraints. This combination can not be solved by **MOSEK**.

`prosta`

Problem status keys

`prosta.unknown`

Unknown problem status.

`prosta.prim_and_dual_feas`

The problem is primal and dual feasible.

`prosta.prim_feas`

The problem is primal feasible.

`prosta.dual_feas`

The problem is dual feasible.

`prosta.prim_infeas`

The problem is primal infeasible.

`prosta.dual_infeas`

The problem is dual infeasible.

`prosta.prim_and_dual_infeas`

The problem is primal and dual infeasible.

`prosta.ill_posed`

The problem is ill-posed. For example, it may be primal and dual feasible but have a positive duality gap.

`prosta.prim_infeas_or_unbounded`

The problem is either primal infeasible or unbounded. This may occur for mixed-integer problems.

`rescodetype`

Response code type

`rescodetype.ok`

The response code is OK.

`rescodetype.wrn`

The response code is a warning.

`rescodetype.trm`

The response code is an optimizer termination status.

`rescodetype.err`

The response code is an error.

`rescodetype.unk`

The response code does not belong to any class.

`scalingtype`

Scaling type

`scalingtype.free`

The optimizer chooses the scaling heuristic.

`scalingtype.none`

No scaling is performed.

`scalingmethod`

Scaling method

`scalingmethod.pow2`

Scales only with power of 2 leaving the mantissa untouched.

`scalingmethod.free`

The optimizer chooses the scaling heuristic.

`sensitivitytype`

Sensitivity types

`sensitivitytype.basis`

Basis sensitivity analysis is performed.

`simseltype`

Simplex selection strategy

`simseltype.free`

The optimizer chooses the pricing strategy.

`simseltype.full`

The optimizer uses full pricing.

`simseltype.ase`

The optimizer uses approximate steepest-edge pricing.

`simseltype.devex`

The optimizer uses devex steepest-edge pricing (or if it is not available an approximate steep-edge selection).

`simseltype.se`

The optimizer uses steepest-edge selection (or if it is not available an approximate steep-edge selection).

`simseltype.partial`

The optimizer uses a partial selection approach. The approach is usually beneficial if the number of variables is much larger than the number of constraints.

`solitem`

Solution items

`solitem.xc`

Solution for the constraints.

`solitem.xx`

Variable solution.

`solitem.y`

Lagrange multipliers for equations.

`solitem.slc`

Lagrange multipliers for lower bounds on the constraints.

`solitem.suc`

Lagrange multipliers for upper bounds on the constraints.

`solitem.slx`

Lagrange multipliers for lower bounds on the variables.

`solitem.sux`

Lagrange multipliers for upper bounds on the variables.

`solitem.snx`

Lagrange multipliers corresponding to the conic constraints on the variables.

`solsta`

Solution status keys

`solsta.unknown`

Status of the solution is unknown.

`solsta.optimal`

The solution is optimal.

`solsta.prim_feas`

The solution is primal feasible.

`solsta.dual_feas`

The solution is dual feasible.

`solsta.prim_and_dual_feas`

The solution is both primal and dual feasible.

`solsta.prim_infeas_cer`

The solution is a certificate of primal infeasibility.

`solsta.dual_infeas_cer`

The solution is a certificate of dual infeasibility.

`solsta.prim_illposed_cer`

The solution is a certificate that the primal problem is illposed.

`solsta.dual_illposed_cer`

The solution is a certificate that the dual problem is illposed.

`solsta.integer_optimal`

The primal solution is integer optimal.

`soltype`

Solution types

`soltype.bas`

The basic solution.

`soltype.itr`

The interior solution.

`soltype.itg`

The integer solution.

`solveform`

Solve primal or dual form

`solveform.free`

The optimizer is free to solve either the primal or the dual problem.

`solveform.primal`

The optimizer should solve the primal problem.

`solveform.dual`

The optimizer should solve the dual problem.

`stakey`

Status keys

`stakey.unk`

The status for the constraint or variable is unknown.

`stakey.bas`

The constraint or variable is in the basis.

`stakey.supbas`

The constraint or variable is super basic.

`stakey.low`

The constraint or variable is at its lower bound.

`stakey.upr`

The constraint or variable is at its upper bound.

`stakey.fix`

The constraint or variable is fixed.

`stakey.inf`

The constraint or variable is infeasible in the bounds.

`startpointtype`

Starting point types

`startpointtype.free`

The starting point is chosen automatically.

`startpointtype.guess`

The optimizer guesses a starting point.

`startpointtype.constant`

The optimizer constructs a starting point by assigning a constant value to all primal and dual variables. This starting point is normally robust.

`streamtype`

Stream types

`streamtype.log`

Log stream. Contains the aggregated contents of all other streams. This means that a message written to any other stream will also be written to this stream.

`streamtype.msg`

Message stream. Log information relating to performance and progress of the optimization is written to this stream.

`streamtype.err`

Error stream. Error messages are written to this stream.

`streamtype.wrn`

Warning stream. Warning messages are written to this stream.

`value`

Integer values

`value.max_str_len`

Maximum string length allowed in **MOSEK**.

`value.license_buffer_length`

The length of a license key buffer.

`variabletype`

Variable types

`variabletype.type_cont`

Is a continuous variable.

`variabletype.type_int`

Is an integer variable.

15.10 Function Types

`callbackfunc`

```
callbackfunc(code, dinf, iinf, liinf) -> stop
```

The progress and information callback function is a user-defined function which will be called by **MOSEK** occasionally during the optimization process. In particular, the callback function is called at the beginning of each iteration in the interior-point optimizer. For the simplex optimizers *iparam.log_sim_freq* controls how frequently the callback is called.

The user *must not* call any **MOSEK** function directly or indirectly from the callback function. The only exception is the possibility to retrieve an integer solution, see *Progress and data callback*.

Parameters

- `code` (*callbackcode*) – Callback code indicating current operation of the solver. (input)
- `dinf` (`float[]`) – Array of double information items. (input)
- `iinf` (`int[]`) – Array of integer information items. (input)
- `liinf` (`int64[]`) – Array of long integer information items. (input)

Return

`stop` (`int`) – Non-zero if the optimizer should be terminated; zero otherwise.

`progresscallbackfunc`

```
progresscallbackfunc(code) -> stop
```

The progress callback function is a user-defined function which will be called by **MOSEK** occasionally during the optimization process. In particular, the callback function is called at the beginning of each iteration in the interior-point optimizer. For the simplex optimizers *iparam.log_sim_freq* controls how frequently the callback is called.

The user *must not* call any **MOSEK** function directly or indirectly from the callback function. If the progress callback function returns a non-zero value, the optimization process is terminated.

Parameters

`code` (*mosek.callbackcode*) – Callback code indicating the current status of the solver. (input)

Return

`stop` (`int`) – Non-zero if the optimizer should be terminated; zero otherwise.

`streamfunc`

```
streamfunc(msg)
```

The message-stream callback function is a user-defined function which can be linked to any of the **MOSEK** streams. Doing so, the function is called whenever **MOSEK** sends a message to the stream.

The user *must not* call any **MOSEK** function directly or indirectly from the callback function.

Parameters
`msg` (`str`) – A string containing the message. (input)

15.11 Supported domains

This section lists the domains supported by **MOSEK**. See Sec. 6 for how to apply domains to specify affine conic constraints (ACCs) and disjunctive constraints (DJCs).

15.11.1 Linear domains

Each linear domain is determined by the dimension n.

- *Task.appendrzerodomain* : the **zero domain**, consisting of the origin $0^n \in \mathbb{R}^n$.
- *Task.appendrplusdomain* : the **nonnegative orthant domain** $\mathbb{R}^n_{\geq 0}$.
- *Task.appendrminusdomain* : the **nonpositive orthant domain** $\mathbb{R}^n_{\leq 0}$.
- *Task.appendrdomain* : the **free domain**, consisting of the whole $\mathbb{R}^n$.

Membership in a linear domain is equivalent to imposing the corresponding set of n linear constraints, for instance $Fx + g \in 0^n$ is equivalent to $Fx + g = 0$ and so on. The free domain imposes no restriction.

15.11.2 Quadratic cone domains

The quadratic domains are determined by the dimension n.

- *Task.appendquadraticconedomain* : the **quadratic cone domain** is the subset of $\mathbb{R}^n$ defined as

$$\mathcal{Q}^n = \left\{ x \in \mathbb{R}^n \ : \ x_1 \geq \sqrt{x_2^2 + \cdots + x_n^2} \right\}.$$

- *Task.appendrquadraticconedomain* : the **rotated quadratic cone domain** is the subset of $\mathbb{R}^n$ defined as

$$\mathcal{Q}_r^n = \left\{ x \in \mathbb{R}^n \ : \ 2x_1 x_2 \geq x_3^2 + \cdots + x_n^2, \ x_1, x_2 \geq 0 \right\}.$$

15.11.3 Exponential cone domains

- *Task.appendprimalexpconedomain* : the **primal exponential cone domain** is the subset of $\mathbb{R}^3$ defined as

$$K_{\exp} = \left\{ (x_1, x_2, x_3) \in \mathbb{R}^3 \ : \ x_1 \geq x_2 \exp(x_3/x_2), \ x_1, x_2 \geq 0 \right\}.$$

- *Task.appenddualexpconedomain* : the **dual exponential cone domain** is the subset of $\mathbb{R}^3$ defined as

$$K_{\exp}^* = \left\{ (x_1, x_2, x_3) \in \mathbb{R}^3 \ : \ x_1 \geq -x_3 \exp(x_2/x_3 - 1), \ x_1 \geq 0, x_3 \leq 0 \right\}.$$

15.11.4 Power cone domains

A power cone domain is determined by the dimension n and a sequence of $1 \leq n_l < n$ positive real numbers (weights) $\alpha_1, \ldots, \alpha_{n_l}$.

- *Task.appendprimalpowerconedomain* : the **primal power cone domain** is the subset of $\mathbb{R}^n$ defined as

$$\mathcal{P}_n^{(\alpha_1,\ldots,\alpha_{n_l})} = \left\{ x \in \mathbb{R}^n \ : \ \prod_{i=1}^{n_l} x_i^{\beta_i} \geq \sqrt{x_{n_l+1}^2 + \cdots + x_n^2}, \ x_1, \ldots, x_{n_l} \geq 0 \right\}.$$

  where β_i are the weights normalized to add up to 1, ie. $\beta_i = \alpha_i / (\sum_j \alpha_j)$ for $i = 1, \ldots, n_l$. The name n_l reads as "n left", the length of the product on the left-hand side of the definition.

- *Task.appenddualpowerconedomain* : the **dual power cone domain** is the subset of $\mathbb{R}^n$ defined as

$$\left(\mathcal{P}_n^{(\alpha_1,\ldots,\alpha_{n_l})}\right)^* = \left\{ x \in \mathbb{R}^n \ : \ \prod_{i=1}^{n_l} \left(\frac{x_i}{\beta_i}\right)^{\beta_i} \geq \sqrt{x_{n_l+1}^2 + \cdots + x_n^2}, \ x_1, \ldots, x_{n_l} \geq 0 \right\}.$$

  where β_i are the weights normalized to add up to 1, ie. $\beta_i = \alpha_i / (\sum_j \alpha_j)$ for $i = 1, \ldots, n_l$. The name n_l reads as "n left", the length of the product on the left-hand side of the definition.

- **Remark:** in MOSEK 9 power cones were available only in the special case with $n_l = 2$ and weights $(\alpha, 1-\alpha)$ for some $0 < \alpha < 1$ specified as cone parameter.

15.11.5 Geometric mean cone domains

A geometric mean cone domain is determined by the dimension n.

- *Task.appendprimalgeomeanconedomain* : the **primal geometric mean cone domain** is the subset of $\mathbb{R}^n$ defined as

$$\mathcal{GM}^n = \left\{ x \in \mathbb{R}^n \ : \ \left(\prod_{i=1}^{n-1} x_i\right)^{1/(n-1)} \geq |x_n|, \ x_1, \ldots, x_{n-1} \geq 0 \right\}.$$

  It is a special case of the primal power cone domain with $n_l = n-1$ and weights $\alpha = (1, \ldots, 1)$.

- *Task.appenddualgeomeanconedomain* : the **dual geometric mean cone domain** is the subset of $\mathbb{R}^n$ defined as

$$(\mathcal{GM}^n)^* = \left\{ x \in \mathbb{R}^n \ : \ (n-1)\left(\prod_{i=1}^{n-1} x_i\right)^{1/(n-1)} \geq |x_n|, \ x_1, \ldots, x_{n-1} \geq 0 \right\}.$$

  It is a special case of the dual power cone domain with $n_l = n-1$ and weights $\alpha = (1, \ldots, 1)$.

15.11.6 Vectorized semidefinite domain

- *Task.appendsvecpsdconedomain* : the **vectorized PSD cone domain** is determined by the dimension n, which must be of the form $n = d(d+1)/2$. Then the domain is defined as

$$\mathcal{S}_+^{d,\mathrm{vec}} = \left\{(x_1, \ldots, x_{d(d+1)/2}) \in \mathbb{R}^n \ : \ \mathrm{sMat}(x) \in \mathcal{S}_+^d\right\},$$

where

$$\mathrm{sMat}(x) = \begin{bmatrix} x_1 & x_2/\sqrt{2} & \cdots & x_d/\sqrt{2} \\ x_2/\sqrt{2} & x_{d+1} & \cdots & x_{2d-1}/\sqrt{2} \\ \cdots & \cdots & \cdots & \cdots \\ x_d/\sqrt{2} & x_{2d-1}/\sqrt{2} & \cdots & x_{d(d+1)/2} \end{bmatrix},$$

or equivalently

$$\mathcal{S}_+^{d,\mathrm{vec}} = \left\{\mathrm{sVec}(X) \ : \ X \in \mathcal{S}_+^d\right\},$$

where

$$\mathrm{sVec}(X) = (X_{11}, \sqrt{2}X_{21}, \ldots, \sqrt{2}X_{d1}, X_{22}, \sqrt{2}X_{32}, \ldots, X_{dd}).$$

In other words, the domain consists of vectorizations of the lower-triangular part of a positive semidefinite matrix, with the non-diagonal elements additionally rescaled.

15.12 Environment variables

This section lists operating system environment variables which can globally affect the behavior of **MOSEK**.

It is recommended that any environment variables are set in the environment *before* launching a process using **MOSEK**, or at the very least before the first time any **MOSEK** library or package is loaded by the process.

- `MOSEKLM_LICENSE_FILE` - location of license file.

  Commonly used to point **MOSEK** to a floating license token server or change the default license file search path. For details see the licensing guide.

- `MOSEK_SYS_NUM_CORES` - set the number of cores.

  When **MOSEK** is loaded it detects the number of cores available on the machine. Setting this environment variable overrides that detection.

  Most users will never need it. Typical applications would be:

  - When **MOSEK** fails to detect the number of cores correctly.
  - To limit the default number of cores available to **MOSEK** in a way transparent to the users.

- `PATH` - system search path.

  In all automated cases (MSI, package managers) the installation process will ensure that the **MOSEK** binaries can be located on runtime, either by adding them to the system search path or via other mechanisms.

  It may be needed to set up by hand for manual, modified or other custom installations.

- `LD_LIBRARY_PATH`, `DYLD_LIBRARY_PATH` - shared objects search path.

  Affects the locations where loader looks for shared libraries. Should never be needed for a correct installation.

- `MIMALLOC_PURGE_DELAY` - mimalloc page release delay.

  Setting it to `0` gives the most aggressive memory release behavior at the cost of speed.

  Most users should never change it. Setting `0` may decrease memory consumption in some special scenarios. Known cases include optimizing very large models many times in a row in the same process. Do not consider it unless memory use becomes an actual issue. Usage at own risk.

Chapter 16

Supported File Formats

MOSEK supports a range of problem and solution formats listed in Table 16.1 and Table 16.2.
The most important are:

- the **Task format**, **MOSEK**'s native binary format which supports all features that **MOSEK** supports. It is the closest possible representation of the internal data in a task and it is ideal for submitting problem data support questions.
- the **PTF format**, **MOSEK**'s human-readable format that supports all linear, conic and mixed-integer features. It is ideal for debugging. It is not an exact copy of all the data in the task, but it contains all information required to reconstruct it, presented in a readable fashion.
- **MPS**, **LP**, **CBF** formats are industry standards, each supporting some limited set of features, and potentially requiring some degree of reformulation during read/write.

Problem formats

Table 16.1: List of supported file formats for optimization problems.

Format Type	Ext.	Binary/Text	LP	QCQO	ACC	SDP	DJC	Sol	Param
LP	lp	plain text	X	X					
MPS	mps	plain text	X	X					
PTF	ptf	plain text	X		X	X	X	X	X
CBF	cbf	plain text	X		X	X			
Task format	task	binary	X	X	X	X	X	X	X
Jtask format	jtask	text/JSON	X	X	X	X	X	X	X
OPF (deprecated for conic problems)	opf	plain text	X	X				X	X

The columns of the table indicate if the specified file format supports:

- LP - linear problems, possibly with integer variables,
- QCQO - quadratic objective or constraints,
- ACC - affine conic constraints,
- SDP - semidefinite cone/variables,
- DJC - disjunctive constraints,
- Sol - solutions,
- Param - optimizer parameters.

Solution formats

Table 16.2: List of supported solution formats.

Format Type	Ext.	Binary/Text	Description
SOL	sol	plain text	Interior Solution
	bas	plain text	Basic Solution
	int	plain text	Integer
Jsol format	jsol	text/JSON	All solutions

Compression

MOSEK supports GZIP and Zstandard compression. Problem files with extension `.gz` (for GZIP) and `.zst` (for Zstandard) are assumed to be compressed when read, and are automatically compressed when written. For example, a file called

```
problem.mps.zst
```

will be considered as a Zstandard compressed MPS file.

16.1 The LP File Format

MOSEK supports the LP file format with some extensions. The LP format is not a completely well-defined standard and hence different optimization packages may interpret the same LP file in slightly different ways. **MOSEK** tries to emulate as closely as possible CPLEX's behavior, but tries to stay backward compatible.

The LP file format can specify problems of the form

$$\begin{array}{lccccc} \text{minimize/maximize} & & & c^T x + \frac{1}{2} q^o(x) & & \\ \text{subject to} & l^c & \leq & Ax + \frac{1}{2} q(x) & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x, \\ & & & x_{\mathcal{J}} \text{ integer}, & & \end{array}$$

where

- $x \in \mathbb{R}^n$ is the vector of decision variables.
- $c \in \mathbb{R}^n$ is the linear term in the objective.
- $q^o :\in \mathbb{R}^n \to \mathbb{R}$ is the quadratic term in the objective where

$$q^o(x) = x^T Q^o x$$

and it is assumed that

$$Q^o = (Q^o)^T.$$

- $A \in \mathbb{R}^{m \times n}$ is the constraint matrix.
- $l^c \in \mathbb{R}^m$ is the lower limit on the activity for the constraints.
- $u^c \in \mathbb{R}^m$ is the upper limit on the activity for the constraints.
- $l^x \in \mathbb{R}^n$ is the lower limit on the activity for the variables.
- $u^x \in \mathbb{R}^n$ is the upper limit on the activity for the variables.
- $q : \mathbb{R}^n \to \mathbb{R}$ is a vector of quadratic functions. Hence,

$$q_i(x) = x^T Q^i x$$

where it is assumed that

$$Q^i = (Q^i)^T.$$

- $\mathcal{J} \subseteq \{1, 2, \ldots, n\}$ is an index set of the integer constrained variables.

16.1.1 File Sections

An LP formatted file contains a number of sections specifying the objective, constraints, variable bounds, and variable types. The section keywords may be any mix of upper and lower case letters.

Objective Function

The first section beginning with one of the keywords

```
max
maximum
maximize
min
minimum
minimize
```

defines the objective sense and the objective function, i.e.

$$c^T x + \frac{1}{2} x^T Q^o x.$$

The objective may be given a name by writing

```
myname:
```

before the expressions.
The objective function contains linear and quadratic terms. The linear terms are written as

```
4 x1 + x2 - 0.1 x3
```

and so forth. The quadratic terms are written in square brackets (`[ ]/2`) and are either squared or multiplied as in the examples

```
x1^2
```

and

```
x1 * x2
```

There may be zero or more pairs of brackets containing quadratic expressions.
An example of an objective section is

```
minimize
myobj: 4 x1 + x2 - 0.1 x3 + [ x1^2 + 2.1 x1 * x2 ]/2
```

Please note that the quadratic expressions are multiplied with $\frac{1}{2}$, so that the above expression means

$$\text{minimize} \quad 4x_1 + x_2 - 0.1 \cdot x_3 + \tfrac{1}{2}(x_1^2 + 2.1 \cdot x_1 \cdot x_2)$$

If the same variable occurs more than once in the linear part, the coefficients are added, so that `4 x1 + 2 x1` is equivalent to `6 x1`. In the quadratic expressions `x1 * x2` is equivalent to `x2 * x1` and, as in the linear part, if the same variables multiplied or squared occur several times their coefficients are added.

Constraints

The second section beginning with one of the keywords

```
subj to
subject to
s.t.
st
```

defines the linear constraint matrix A and the quadratic matrices Q^i.

A constraint contains a name (optional), expressions adhering to the same rules as in the objective and a bound:

```
subject to
con1: x1 + x2 + [ x3^2 ]/2 <= 5.1
```

The bound type (here `<=`) may be any of `<`, `<=`, `=`, `>`, `>=` (`<` and `<=` mean the same), and the bound may be any number.

Ranged constraints cannot be written in LP format, and have to be split into a separate upper and lower bound.

Bounds

Bounds on the variables can be specified in the bound section beginning with one of the keywords

```
bound
bounds
```

The bounds section is optional but should, if present, follow the `subject to` section. All variables listed in the bounds section must occur in either the objective or a constraint.

The default lower and upper bounds are 0 and $+\infty$. A variable may be declared free with the keyword `free`, which means that the lower bound is $-\infty$ and the upper bound is $+\infty$. Furthermore it may be assigned a finite lower and upper bound. The bound definitions for a given variable may be written in one or two lines, and bounds can be any number or $\pm\infty$ (written as `+inf`/`-inf`/`+infinity`/`-infinity`) as in the example

```
bounds
x1 free
x2 <= 5
0.1 <= x2
x3 = 42
2 <= x4 < +inf
```

Variable Types

The final two sections are optional and must begin with one of the keywords

```
bin
binaries
binary
```

and

```
gen
general
```

Under `general` all integer variables are listed, and under `binary` all binary (integer variables with bounds 0 and 1) are listed:

```
general
x1 x2
```

(continues on next page)

(continued from previous page)

```
binary
x3 x4
```

Again, all variables listed in the binary or general sections must occur in either the objective or a constraint.

Terminating Section

Finally, an LP formatted file must be terminated with the keyword

```
end
```

16.1.2 LP File Examples

Linear example `lo1.lp`

```
\ File: lo1.lp
maximize
obj: 3 x1 + x2 + 5 x3 + x4
subject to
c1:  3 x1 + x2 + 2 x3 = 30
c2:  2 x1 + x2 + 3 x3 + x4 >= 15
c3:  2 x2 + 3 x4 <= 25
bounds
 0 <= x1 <= +infinity
 0 <= x2 <= 10
 0 <= x3 <= +infinity
 0 <= x4 <= +infinity
end
```

Mixed integer example `milo1.lp`

```
maximize
obj: x1 + 6.4e-01 x2
subject to
c1:  5e+01 x1 + 3.1e+01 x2 <= 2.5e+02
c2:  3e+00 x1 - 2e+00 x2 >= -4e+00
bounds
 0 <= x1 <= +infinity
 0 <= x2 <= +infinity
general
 x1 x2
end
```

16.1.3 LP Format peculiarities

Comments

Anything on a line after a \ is ignored and is treated as a comment.

Names

A name for an objective, a constraint or a variable may contain the letters `a-z`, `A-Z`, the digits `0-9` and the characters

```
!"#$%&()/,.;?@_'`|~
```

The first character in a name must not be a number, a period or the letter `e` or `E`. Keywords must not be used as names.

MOSEK accepts any character as valid for names, except `\0`. A name that is not allowed in LP file will be changed and a warning will be issued.

The algorithm for making names LP valid works as follows: The name is interpreted as an `utf-8` string. For a Unicode character `c`:

- If `c==_` (underscore), the output is `__` (two underscores).
- If `c` is a valid LP name character, the output is just `c`.
- If `c` is another character in the ASCII range, the output is `_XX`, where `XX` is the hexadecimal code for the character.
- If `c` is a character in the range *127-65535*, the output is `_uXXXX`, where `XXXX` is the hexadecimal code for the character.
- If `c` is a character above 65535, the output is `_UXXXXXXXX`, where `XXXXXXXX` is the hexadecimal code for the character.

Invalid `utf-8` substrings are escaped as `_XX'`, and if a name starts with a period, `e` or `E`, that character is escaped as `_XX`.

Variable Bounds

Specifying several upper or lower bounds on one variable is possible but **MOSEK** uses only the tightest bounds. If a variable is fixed (with `=`), then it is considered the tightest bound.

16.2 The MPS File Format

MOSEK supports the standard MPS format with some extensions. For a detailed description of the MPS format see the book by Nazareth [Naz87].

16.2.1 MPS File Structure

The version of the MPS format supported by **MOSEK** allows specification of an optimization problem of the form

$$\begin{array}{rccccl} \text{maximize/minimize} & & c^T x + q_0(x) & & & \\ l^c & \leq & Ax + q(x) & \leq & u^c, & \\ l^x & \leq & x & \leq & u^x, & \\ & & x \in \mathcal{K}, & & & \\ & & x_{\mathcal{J}} \ \text{integer}, & & & \end{array} \tag{16.1}$$

where

- $x \in \mathbb{R}^n$ is the vector of decision variables.
- $A \in \mathbb{R}^{m \times n}$ is the constraint matrix.
- $l^c \in \mathbb{R}^m$ is the lower limit on the activity for the constraints.
- $u^c \in \mathbb{R}^m$ is the upper limit on the activity for the constraints.
- $l^x \in \mathbb{R}^n$ is the lower limit on the activity for the variables.
- $u^x \in \mathbb{R}^n$ is the upper limit on the activity for the variables.

- $q : \mathbb{R}^n \to \mathbb{R}$ is a vector of quadratic functions. Hence,

$$q_i(x) = \frac{1}{2} x^T Q^i x$$

  where it is assumed that $Q^i = (Q^i)^T$. Please note the explicit $\frac{1}{2}$ in the quadratic term and that Q^i is required to be symmetric. The same applies to q_0.

- $\mathcal{K}$ is a convex cone.
- $\mathcal{J} \subseteq \{1, 2, \ldots, n\}$ is an index set of the integer-constrained variables.
- c is the vector of objective coefficients.

An MPS file with one row and one column can be illustrated like this:

```
*        1         2         3         4         5         6
*23456789012345678901234567890123456789012345678901234567890
NAME          [name]
OBJSENSE
    [objsense]
OBJNAME       [objname]
ROWS
 ?  [cname1]
COLUMNS
    [vname1]  [cname1]  [value1]       [cname2]  [value2]
RHS
    [name]    [cname1]  [value1]       [cname2]  [value2]
RANGES
    [name]    [cname1]  [value1]       [cname2]  [value2]
QSECTION      [cname1]
    [vname1]  [vname2]  [value1]       [vname3]  [value2]
QMATRIX
    [vname1]  [vname2]  [value1]
QUADOBJ
    [vname1]  [vname2]  [value1]
QCMATRIX      [cname1]
    [vname1]  [vname2]  [value1]
BOUNDS
 ?? [name]    [vname1]  [value1]
CSECTION      [kname1]  [value1]       [ktype]
    [vname1]
ENDATA
```

Here the names in capitals are keywords of the MPS format and names in brackets are custom defined names or values. A couple of notes on the structure:

- Fields: All items surrounded by brackets appear in *fields*. The fields named "`valueN`" are numerical values. Hence, they must have the format

  ```
  [+|-]XXXXXXX.XXXXXX[[e|E][+|-]XXX]
  ```

  where

  ```
  X = [0|1|2|3|4|5|6|7|8|9].
  ```

- Sections: The MPS file consists of several sections where the names in capitals indicate the beginning of a new section. For example, `COLUMNS` denotes the beginning of the columns section.
- Comments: Lines starting with an `*` are comment lines and are ignored by **MOSEK**.
- Keys: The question marks represent keys to be specified later.

- Extensions: The sections `QSECTION` and `CSECTION` are specific **MOSEK** extensions of the MPS format. The sections `QMATRIX`, `QUADOBJ` and `QCMATRIX` are included for sake of compatibility with other vendors extensions to the MPS format.
- The standard MPS format is a fixed format, i.e. everything in the MPS file must be within certain fixed positions. **MOSEK** also supports a *free format*. See Sec. 16.2.5 for details.

Linear example `lo1.mps`

A concrete example of a MPS file is presented below:

```
* File: lo1.mps
NAME          lo1
OBJSENSE
    MAX
ROWS
 N  obj
 E  c1
 G  c2
 L  c3
COLUMNS
    x1        obj       3
    x1        c1        3
    x1        c2        2
    x2        obj       1
    x2        c1        1
    x2        c2        1
    x2        c3        2
    x3        obj       5
    x3        c1        2
    x3        c2        3
    x4        obj       1
    x4        c2        1
    x4        c3        3
RHS
    rhs       c1        30
    rhs       c2        15
    rhs       c3        25
RANGES
BOUNDS
 UP bound     x2        10
ENDATA
```

Subsequently each individual section in the MPS format is discussed.

NAME (optional)

In this section a name ([name]) is assigned to the problem.

OBJSENSE (optional)

This is an optional section that can be used to specify the sense of the objective function. The OBJSENSE section contains one line at most which can be one of the following:

```
MIN
MINIMIZE
MAX
MAXIMIZE
```

It should be obvious what the implication is of each of these four lines.

OBJNAME (optional)

This is an optional section that can be used to specify the name of the row that is used as objective function. objname should be a valid row name.

ROWS

A record in the ROWS section has the form

```
?  [cname1]
```

where the requirements for the fields are as follows:

Field	Starting Position	Max Width	required	Description
?	2	1	Yes	Constraint key
[cname1]	5	8	Yes	Constraint name

Hence, in this section each constraint is assigned a unique name denoted by [cname1]. Please note that [cname1] starts in position 5 and the field can be at most 8 characters wide. An initial key ? must be present to specify the type of the constraint. The key can have values E, G, L, or N with the following interpretation:

Constraint type	l_i^c	u_i^c
E (equal)	finite	$= l_i^c$
G (greater)	finite	∞
L (lower)	$-\infty$	finite
N (none)	$-\infty$	∞

In the MPS format the objective vector is not specified explicitly, but one of the constraints having the key N will be used as the objective vector c . In general, if multiple N type constraints are specified, then the first will be used as the objective vector c, unless something else was specified in the section OBJNAME.

COLUMNS

In this section the elements of A are specified using one or more records having the form:

```
[vname1]   [cname1]     [value1]      [cname2]   [value2]
```

where the requirements for each field are as follows:

Field	Starting Position	Max Width	required	Description
[vname1]	5	8	Yes	Variable name
[cname1]	15	8	Yes	Constraint name
[value1]	25	12	Yes	Numerical value
[cname2]	40	8	No	Constraint name
[value2]	50	12	No	Numerical value

Hence, a record specifies one or two elements a_{ij} of A using the principle that [vname1] and [cname1] determines j and i respectively. Please note that [cname1] must be a constraint name specified in the ROWS section. Finally, [value1] denotes the numerical value of a_{ij} . Another optional element is specified by [cname2], and [value2] for the variable specified by [vname1]. Some important comments are:

- All elements belonging to one variable must be grouped together.
- Zero elements of A should not be specified.
- At least one element for each variable should be specified.

RHS (optional)

A record in this section has the format

```
[name]    [cname1]    [value1]    [cname2]  [value2]
```

where the requirements for each field are as follows:

Field	Starting Position	Max Width	required	Description
[name]	5	8	Yes	Name of the RHS vector
[cname1]	15	8	Yes	Constraint name
[value1]	25	12	Yes	Numerical value
[cname2]	40	8	No	Constraint name
[value2]	50	12	No	Numerical value

The interpretation of a record is that [name] is the name of the RHS vector to be specified. In general, several vectors can be specified. [cname1] denotes a constraint name previously specified in the ROWS section. Now, assume that this name has been assigned to the i -h constraint and v_1 denotes the value specified by [value1], then the interpretation of v_1 is:

Constraint	l_i^c	u_i^c
E	v_1	v_1
G	v_1	
L		v_1
N		

An optional second element is specified by [cname2] and [value2] and is interpreted in the same way. Please note that it is not necessary to specify zero elements, because elements are assumed to be zero.

RANGES (optional)

A record in this section has the form

```
[name]    [cname1]    [value1]    [cname2]  [value2]
```

where the requirements for each fields are as follows:

Field	Starting Position	Max Width	required	Description
[name]	5	8	Yes	Name of the RANGE vector
[cname1]	15	8	Yes	Constraint name
[value1]	25	12	Yes	Numerical value
[cname2]	40	8	No	Constraint name
[value2]	50	12	No	Numerical value

The records in this section are used to modify the bound vectors for the constraints, i.e. the values in l^c and u^c . A record has the following interpretation:[name] is the name of the RANGE vector and [cname1] is a valid constraint name. Assume that [cname1] is assigned to the i-th constraint and let v_1 be the value specified by [value1], then a record has the interpretation:

Constraint type	Sign of v_1	l_i^c	u_i^c
E	$-$	$u_i^c + v_1$	
E	$+$		$l_i^c + v_1$
G	$-$ or $+$		$l_i^c + \|v_1\|$
L	$-$ or $+$	$u_i^c - \|v_1\|$	
N			

Another constraint bound can optionally be modified using [cname2] and [value2] the same way.

QSECTION (optional)

Within the QSECTION the label [cname1] must be a constraint name previously specified in the ROWS section. The label [cname1] denotes the constraint to which the quadratic terms belong. A record in the QSECTION has the form

```
[vname1]   [vname2]    [value1]    [vname3]   [value2]
```

where the requirements for each field are:

Field	Starting Position	Max Width	required	Description
[vname1]	5	8	Yes	Variable name
[vname2]	15	8	Yes	Variable name
[value1]	25	12	Yes	Numerical value
[vname3]	40	8	No	Variable name
[value2]	50	12	No	Numerical value

A record specifies one or two elements in the lower triangular part of the Q^i matrix where [cname1] specifies the i . Hence, if the names [vname1] and [vname2] have been assigned to the k-th and j-th variable, then Q^i_{kj} is assigned the value given by [value1] An optional second element is specified in the same way by the fields [vname1], [vname3], and [value2].

The example

$$\begin{array}{lcll} \text{minimize} & -x_2 + \frac{1}{2}(2x_1^2 - 2x_1x_3 + 0.2x_2^2 + 2x_3^2) & & \\ \text{subject to} & x_1 + x_2 + x_3 & \geq & 1, \\ & x \geq 0 & & \end{array}$$

has the following MPS file representation

```
* File: qo1.mps
NAME          qo1
ROWS
 N  obj
 G  c1
COLUMNS
    x1        c1        1.0
    x2        obj       -1.0
    x2        c1        1.0
    x3        c1        1.0
RHS
    rhs       c1        1.0
QSECTION      obj
    x1        x1        2.0
    x1        x3        -1.0
    x2        x2        0.2
    x3        x3        2.0
ENDATA
```

Regarding the QSECTIONs please note that:

- Only one QSECTION is allowed for each constraint.
- The QSECTIONs can appear in an arbitrary order after the COLUMNS section.
- All variable names occurring in the QSECTION must already be specified in the COLUMNS section.
- All entries specified in a QSECTION are assumed to belong to the lower triangular part of the quadratic term of Q .

QMATRIX/QUADOBJ (optional)

The QMATRIX and QUADOBJ sections allow to define the quadratic term of the objective function. They differ in how the quadratic term of the objective function is stored:

- QMATRIX stores all the nonzeros coefficients, without taking advantage of the symmetry of the Q matrix.
- QUADOBJ stores the upper diagonal nonzero elements of the Q matrix.

A record in both sections has the form:

```
[vname1]   [vname2]     [value1]
```

where the requirements for each field are:

Field	Starting Position	Max Width	required	Description
[vname1]	5	8	Yes	Variable name
[vname2]	15	8	Yes	Variable name
[value1]	25	12	Yes	Numerical value

A record specifies one elements of the Q matrix in the objective function . Hence, if the names [vname1] and [vname2] have been assigned to the k-th and j-th variable, then Q_{kj} is assigned the value given by [value1]. Note that a line must appear for each off-diagonal coefficient if using a QMATRIX section, while only one entry is required in a QUADOBJ section. The quadratic part of the objective function will be evaluated as $1/2x^TQx$.

The example

$$\begin{array}{lcl} \text{minimize} & -x_2 + \frac{1}{2}(2x_1^2 - 2x_1x_3 + 0.2x_2^2 + 2x_3^2) & \\ \text{subject to} & x_1 + x_2 + x_3 & \geq \quad 1, \\ & x \geq 0 & \end{array}$$

has the following MPS file representation using QMATRIX

```
* File: qo1_matrix.mps
NAME          qo1_qmatrix
ROWS
 N  obj
 G  c1
COLUMNS
    x1        c1        1.0
    x2        obj       -1.0
    x2        c1        1.0
    x3        c1        1.0
RHS
    rhs       c1        1.0
QMATRIX
    x1        x1        2.0
    x1        x3        -1.0
    x3        x1        -1.0
    x2        x2        0.2
    x3        x3        2.0
ENDATA
```

or the following using `QUADOBJ`

```
* File: qo1_quadobj.mps
NAME          qo1_quadobj
ROWS
 N  obj
 G  c1
COLUMNS
    x1        c1        1.0
    x2        obj       -1.0
    x2        c1        1.0
    x3        c1        1.0
RHS
    rhs       c1        1.0
QUADOBJ
    x1        x1        2.0
    x1        x3        -1.0
    x2        x2        0.2
    x3        x3        2.0
ENDATA
```

Please also note that:

- A `QMATRIX`/`QUADOBJ` section can appear in an arbitrary order after the `COLUMNS` section.
- All variable names occurring in the `QMATRIX`/`QUADOBJ` section must already be specified in the `COLUMNS` section.

`QCMATRIX` (optional)

A `QCMATRIX` section allows to specify the quadratic part of a given constraint. Within the `QCMATRIX` the label `[cname1]` must be a constraint name previously specified in the `ROWS` section. The label `[cname1]` denotes the constraint to which the quadratic term belongs. A record in the `QSECTION` has the form

```
[vname1]  [vname2]    [value1]
```

where the requirements for each field are:

Field	Starting Position	Max Width	required	Description
[vname1]	5	8	Yes	Variable name
[vname2]	15	8	Yes	Variable name
[value1]	25	12	Yes	Numerical value

A record specifies an entry of the Q^i matrix where [cname1] specifies the i. Hence, if the names [vname1] and [vname2] have been assigned to the k-th and j-th variable, then Q^i_{kj} is assigned the value given by [value1]. Moreover, the quadratic term is represented as $1/2x^TQx$.

The example

$$\begin{array}{lcll} \text{minimize} & x_2 & & \\ \text{subject to} & x_1 + x_2 + x_3 & \geq & 1, \\ & \frac{1}{2}(-2x_1x_3 + 0.2x_2^2 + 2x_3^2) & \leq & 10, \\ & x \geq 0 & & \end{array}$$

has the following MPS file representation

```
* File: qo1.mps
NAME          qo1
ROWS
 N  obj
 G  c1
 L  q1
COLUMNS
    x1        c1        1.0
    x2        obj       -1.0
    x2        c1        1.0
    x3        c1        1.0
RHS
    rhs       c1        1.0
    rhs       q1        10.0
QCMATRIX      q1
    x1        x1        2.0
    x1        x3        -1.0
    x3        x1        -1.0
    x2        x2        0.2
    x3        x3        2.0
ENDATA
```

Regarding the QCMATRIXs please note that:

- Only one QCMATRIX is allowed for each constraint.
- The QCMATRIXs can appear in an arbitrary order after the COLUMNS section.
- All variable names occurring in the QSECTION must already be specified in the COLUMNS section.
- QCMATRIX does not exploit the symmetry of Q: an off-diagonal entry (i, j) should appear twice.

BOUNDS (optional)

In the BOUNDS section changes to the default bounds vectors l^x and u^x are specified. The default bounds vectors are $l^x = 0$ and $u^x = \infty$. Moreover, it is possible to specify several sets of bound vectors. A record in this section has the form

```
?? [name]     [vname1]     [value1]
```

where the requirements for each field are:

Field	Starting Position	Max Width	Required	Description
??	2	2	Yes	Bound key
[name]	5	8	Yes	Name of the BOUNDS vector
[vname1]	15	8	Yes	Variable name
[value1]	25	12	No	Numerical value

Hence, a record in the BOUNDS section has the following interpretation: [name] is the name of the bound vector and [vname1] is the name of the variable for which the bounds are modified by the record. ?? and [value1] are used to modify the bound vectors according to the following table:

??	l_j^x	u_j^x	Made integer (added to $\mathcal{J}$)
FR	$-\infty$	∞	No
FX	v_1	v_1	No
LO	v_1	unchanged	No
MI	$-\infty$	unchanged	No
PL	unchanged	∞	No
UP	unchanged	v_1	No
BV	0	1	Yes
LI	$\lceil v_1 \rceil$	unchanged	Yes
UI	unchanged	$\lfloor v_1 \rfloor$	Yes

Here v_1 is the value specified by [value1].

CSECTION (optional)

The purpose of the CSECTION is to specify the conic constraint

$$x \in \mathcal{K}$$

in (16.1). It is assumed that $\mathcal{K}$ satisfies the following requirements. Let

$$x^t \in \mathbb{R}^{n^t}, \quad t = 1, \ldots, k$$

be vectors comprised of parts of the decision variables x so that each decision variable is a member of exactly **one** vector x^t, for example

$$x^1 = \begin{bmatrix} x_1 \\ x_4 \\ x_7 \end{bmatrix} \quad \text{and} \quad x^2 = \begin{bmatrix} x_6 \\ x_5 \\ x_3 \\ x_2 \end{bmatrix}.$$

Next define

$$\mathcal{K} := \left\{ x \in \mathbb{R}^n : \quad x^t \in \mathcal{K}_t, \quad t = 1, \ldots, k \right\}$$

where $\mathcal{K}_t$ must have one of the following forms:

- $\mathbb{R}$ set:

$$\mathcal{K}_t = \mathbb{R}^{n^t}.$$

- Zero cone:

$$\mathcal{K}_t = \{0\} \subseteq \mathbb{R}^{n^t}. \tag{16.2}$$

- Quadratic cone:

$$\mathcal{K}_t = \left\{ x \in \mathbb{R}^{n^t} : x_1 \geq \sqrt{\sum_{j=2}^{n^t} x_j^2} \right\}. \tag{16.3}$$

- Rotated quadratic cone:

$$\mathcal{K}_t = \left\{ x \in \mathbb{R}^{n^t} : 2x_1 x_2 \geq \sum_{j=3}^{n^t} x_j^2, \quad x_1, x_2 \geq 0 \right\}. \tag{16.4}$$

- Primal exponential cone:

$$\mathcal{K}_t = \left\{ x \in \mathbb{R}^3 : x_1 \geq x_2 \exp(x_3/x_2), \quad x_1, x_2 \geq 0 \right\}. \tag{16.5}$$

- Primal power cone (with parameter $0 < \alpha < 1$):

$$\mathcal{K}_t = \left\{ x \in \mathbb{R}^{n^t} : x_1^{\alpha} x_2^{1-\alpha} \geq \sqrt{\sum_{j=3}^{n^t} x_j^2}, \quad x_1, x_2 \geq 0 \right\}. \tag{16.6}$$

- Dual exponential cone:

$$\mathcal{K}_t = \left\{ x \in \mathbb{R}^3 : x_1 \geq -x_3 e^{-1} \exp(x_2/x_3), \quad x_3 \leq 0, x_1 \geq 0 \right\}. \tag{16.7}$$

- Dual power cone (with parameter $0 < \alpha < 1$):

$$\mathcal{K}_t = \left\{ x \in \mathbb{R}^{n^t} : \left(\frac{x_1}{\alpha}\right)^{\alpha} \left(\frac{x_2}{1-\alpha}\right)^{1-\alpha} \geq \sqrt{\sum_{j=3}^{n^t} x_j^2}, \quad x_1, x_2 \geq 0 \right\}. \tag{16.8}$$

In general, membership in the $\mathbb{R}$ set is not specified. If a variable is not a member of any other cone then it is assumed to be a member of the $\mathbb{R}$ cone.

Next, let us study an example. Assume that the power cone

$$x_4^{1/3} x_5^{2/3} \geq |x_8|$$

and the rotated quadratic cone

$$2x_3 x_7 \geq x_1^2 + x_0^2, \quad x_3, x_7 \geq 0,$$

should be specified in the MPS file. One `CSECTION` is required for each cone and they are specified as follows:

```
*        1         2         3         4         5         6
*2345678901234567890123456789012345678901234567890123456789012345678901234567890
CSECTION      konea     3e-1            PPOW
x4
x5
x8
CSECTION      koneb     0.0             RQUAD
x7
x3
x1
x0
```

In general, a `CSECTION` header has the format

```
CSECTION      [kname1]     [value1]     [ktype]
```

where the requirements for each field are as follows:

Field	Starting Position	Max Width	Required	Description
`[kname1]`	15	8	Yes	Name of the cone
`[value1]`	25	12	No	Cone parameter
`[ktype]`	40		Yes	Type of the cone.

The possible cone type keys are:

[ktype]	Members	[value1]	Interpretation.
ZERO	≥ 0	unused	Zero cone (16.2).
QUAD	≥ 1	unused	Quadratic cone (16.3).
RQUAD	≥ 2	unused	Rotated quadratic cone (16.4).
PEXP	3	unused	Primal exponential cone (16.5).
PPOW	≥ 2	α	Primal power cone (16.6).
DEXP	3	unused	Dual exponential cone (16.7).
DPOW	≥ 2	α	Dual power cone (16.8).

A record in the CSECTION has the format

```
[vname1]
```

where the requirements for each field are

Field	Starting Position	Max Width	required	Description
[vname1]	5	8	Yes	A valid variable name

A variable must occur in at most one CSECTION.

ENDATA

This keyword denotes the end of the MPS file.

16.2.2 Integer Variables

Using special bound keys in the BOUNDS section it is possible to specify that some or all of the variables should be integer-constrained i.e. be members of $\mathcal{J}$. However, an alternative method is available. This method is available only for backward compatibility and we recommend that it is not used. This method requires that markers are placed in the COLUMNS section as in the example:

```
COLUMNS
x1        obj       -10.0          c1        0.7
x1        c2        0.5            c3        1.0
x1        c4        0.1
* Start of integer-constrained variables.
MARK000   'MARKER'                 'INTORG'
x2        obj       -9.0           c1        1.0
x2        c2        0.8333333333   c3        0.66666667
x2        c4        0.25
x3        obj       1.0            c6        2.0
MARK001   'MARKER'                 'INTEND'
* End of integer-constrained variables.
```

Please note that special marker lines are used to indicate the start and the end of the integer variables. Furthermore be aware of the following

- All variables between the markers are assigned a default lower bound of 0 and a default upper bound of 1. **This may not be what is intended.** If it is not intended, the correct bounds should be defined in the BOUNDS section of the MPS formatted file.
- **MOSEK** ignores field 1, i.e. MARK0001 and MARK001, however, other optimization systems require them.
- Field 2, i.e. MARKER, must be specified including the single quotes. This implies that no row can be assigned the name MARKER.
- Field 3 is ignored and should be left blank.

- Field 4, i.e. `INTORG` and `INTEND`, must be specified.
- It is possible to specify several such integer marker sections within the `COLUMNS` section.

16.2.3 General Limitations

- An MPS file should be an ASCII file.

16.2.4 Interpretation of the MPS Format

Several issues related to the MPS format are not well-defined by the industry standard. However, **MOSEK** uses the following interpretation:

- If a matrix element in the `COLUMNS` section is specified multiple times, then the multiple entries are added together.
- If a matrix element in a `QSECTION` section is specified multiple times, then the multiple entries are added together.

16.2.5 The Free MPS Format

MOSEK supports a free format variation of the MPS format. The free format is similar to the MPS file format but less restrictive, e.g. it allows longer names. However, a name must not contain any blanks.

Moreover, by default a line in the MPS file must not contain more than 1024 characters. By modifying the parameter *iparam.read_mps_width* an arbitrary large line width will be accepted.

The free MPS format is default. To change to the strict and other formats use the parameter *iparam.read_mps_format*.

> **Warning:** This file format is to a large extent deprecated. While it can still be used for linear and quadratic problems, for conic problems the Sec. 16.5 is recommended.

16.3 The OPF Format

The *Optimization Problem Format (OPF)* is an alternative to LP and MPS files for specifying optimization problems. It is row-oriented, inspired by the CPLEX LP format.

Apart from containing objective, constraints, bounds etc. it may contain complete or partial solutions, comments and extra information relevant for solving the problem. It is designed to be easily read and modified by hand and to be forward compatible with possible future extensions.

Intended use

The OPF file format is meant to replace several other files:

- The LP file format: Any problem that can be written as an LP file can be written as an OPF file too; furthermore it naturally accommodates ranged constraints and variables as well as arbitrary characters in names, fixed expressions in the objective, empty constraints, and conic constraints.
- Parameter files: It is possible to specify integer, double and string parameters along with the problem (or in a separate OPF file).
- Solution files: It is possible to store a full or a partial solution in an OPF file and later reload it.

16.3.1 The File Format

The format uses tags to structure data. A simple example with the basic sections may look like this:

```
[comment]
This is a comment. You may write almost anything here...
[/comment]

# This is a single-line comment.

[objective min 'myobj']
x + 3 y + x^2 + 3 y^2 + z + 1
[/objective]

[constraints]
[con 'con01'] 4 <= x + y  [/con]
[/constraints]

[bounds]
[b] -10 <= x,y <= 10  [/b]

[cone quad] x,y,z [/cone]
[/bounds]
```

A scope is opened by a tag of the form `[tag]` and closed by a tag of the form `[/tag]`. An opening tag may accept a list of unnamed and named arguments, for examples:

```
[tag value] tag with one unnamed argument [/tag]
[tag arg=value] tag with one named argument [/tag]
```

Unnamed arguments are identified by their order, while named arguments may appear in any order, but never before an unnamed argument. The `value` can be a quoted, single-quoted or double-quoted text string, i.e.

```
[tag 'value']     single-quoted value [/tag]
[tag arg='value'] single-quoted value [/tag]
[tag "value"]     double-quoted value [/tag]
[tag arg="value"] double-quoted value [/tag]
```

16.3.2 Sections

The recognized tags are

[comment]

A comment section. This can contain *almost* any text: Between single quotes (') or double quotes (") any text may appear. Outside quotes the markup characters ([and]) must be prefixed by backslashes. Both single and double quotes may appear alone or inside a pair of quotes if it is prefixed by a backslash.

[objective]

The objective function: This accepts one or two parameters, where the first one (in the above example `min`) is either `min` or `max` (regardless of case) and defines the objective sense, and the second one (above `myobj`), if present, is the objective name. The section may contain linear and quadratic expressions.

If several objectives are specified, all but the last are ignored.

[constraints]

This does not directly contain any data, but may contain subsections `con` defining a linear constraint.

[con]

Defines a single constraint; if an argument is present (`[con NAME]`) this is used as the name of the constraint, otherwise it is given a null-name. The section contains a constraint definition written as linear and quadratic expressions with a lower bound, an upper bound, with both or with an equality. Examples:

```
[constraints]
[con 'con1'] 0 <= x + y       [/con]
[con 'con2'] 0 >= x + y       [/con]
[con 'con3'] 0 <= x + y <= 10 [/con]
[con 'con4']      x + y  = 10 [/con]
[/constraints]
```

Constraint names are unique. If a constraint is specified which has the same name as a previously defined constraint, the new constraint replaces the existing one.

[bounds]

This does not directly contain any data, but may contain subsections `b` (linear bounds on variables) and `cone` (cones).

[b]

Bound definition on one or several variables separated by comma (`,`). An upper or lower bound on a variable replaces any earlier defined bound on that variable. If only one bound (upper or lower) is given only this bound is replaced. This means that upper and lower bounds can be specified separately. So the OPF bound definition:

```
[b]  x,y >= -10  [/b]
[b]  x,y <= 10   [/b]
```

results in the bound $-10 \leq x, y \leq 10$.

`[cone]`

Specifies a cone. A cone is defined as a sequence of variables which belong to a single unique cone. The supported cone types are:

- `quad`: a quadratic cone of n variables $x_1, \ldots, x_n$ defines a constraint of the form

$$x_1^2 \geq \sum_{i=2}^{n} x_i^2, \quad x_1 \geq 0.$$

- `rquad`: a rotated quadratic cone of n variables $x_1, \ldots, x_n$ defines a constraint of the form

$$2x_1 x_2 \geq \sum_{i=3}^{n} x_i^2, \quad x_1, x_2 \geq 0.$$

- `pexp`: primal exponential cone of 3 variables x_1, x_2, x_3 defines a constraint of the form

$$x_1 \geq x_2 \exp(x_3/x_2), \quad x_1, x_2 \geq 0.$$

- `ppow` with parameter $0 < \alpha < 1$: primal power cone of n variables $x_1, \ldots, x_n$ defines a constraint of the form

$$x_1^{\alpha} x_2^{1-\alpha} \geq \sqrt{\sum_{j=3}^{n} x_j^2}, \quad x_1, x_2 \geq 0.$$

- `dexp`: dual exponential cone of 3 variables x_1, x_2, x_3 defines a constraint of the form

$$x_1 \geq -x_3 e^{-1} \exp(x_2/x_3), \quad x_3 \leq 0, x_1 \geq 0.$$

- `dpow` with parameter $0 < \alpha < 1$: dual power cone of n variables $x_1, \ldots, x_n$ defines a constraint of the form

$$\left(\frac{x_1}{\alpha}\right)^{\alpha} \left(\frac{x_2}{1-\alpha}\right)^{1-\alpha} \geq \sqrt{\sum_{j=3}^{n^t} x_j^2}, \quad x_1, x_2 \geq 0.$$

- `zero`: zero cone of n variables $x_1, \ldots, x_n$ defines a constraint of the form

$$x_1 = \cdots = x_n = 0$$

A `[bounds]`-section example:

```
[bounds]
[b]  0 <= x,y <= 10  [/b] # ranged bound
[b] 10 >= x,y >=  0  [/b] # ranged bound
[b]  0 <= x,y <= inf [/b] # using inf
[b]       x,y free   [/b] # free variables
# Let (x,y,z,w) belong to the cone K
[cone rquad] x,y,z,w  [/cone] # rotated quadratic cone
[cone ppow '3e-01' 'a'] x1, x2, x3 [/cone] # power cone with alpha=1/3 and name 'a'
[/bounds]
```

By default all variables are free.

[variables]

This defines an ordering of variables as they should appear in the problem. This is simply a space-separated list of variable names.

[integer]

This contains a space-separated list of variables and defines the constraint that the listed variables must be integer-valued.

[hints]

This may contain only non-essential data; for example estimates of the number of variables, constraints and non-zeros. Placed before all other sections containing data this may reduce the time spent reading the file.

In the `hints` section, any subsection which is not recognized by **MOSEK** is simply ignored. In this section a hint is defined as follows:

```
[hint ITEM] value [/hint]
```

The hints recognized by **MOSEK** are:

- `numvar` (number of variables),
- `numcon` (number of linear/quadratic constraints),
- `numanz` (number of linear non-zeros in constraints),
- `numqnz` (number of quadratic non-zeros in constraints).

[solutions]

This section can contain a set of full or partial solutions to a problem. Each solution must be specified using a `[solution]`-section, i.e.

```
[solutions]
[solution]...[/solution] #solution 1
[solution]...[/solution] #solution 2
#other solutions....
[solution]...[/solution] #solution n
[/solutions]
```

The syntax of a `[solution]`-section is the following:

```
[solution SOLTYPE status=STATUS]...[/solution]
```

where `SOLTYPE` is one of the strings

- `interior`, a non-basic solution,
- `basic`, a basic solution,
- `integer`, an integer solution,

and `STATUS` is one of the strings

- `UNKNOWN`,
- `OPTIMAL`,
- `INTEGER_OPTIMAL`,
- `PRIM_FEAS`,
- `DUAL_FEAS`,
- `PRIM_AND_DUAL_FEAS`,

- NEAR_OPTIMAL,
- NEAR_PRIM_FEAS,
- NEAR_DUAL_FEAS,
- NEAR_PRIM_AND_DUAL_FEAS,
- PRIM_INFEAS_CER,
- DUAL_INFEAS_CER,
- NEAR_PRIM_INFEAS_CER,
- NEAR_DUAL_INFEAS_CER,
- NEAR_INTEGER_OPTIMAL.

Most of these values are irrelevant for input solutions; when constructing a solution for simplex hot-start or an initial solution for a mixed integer problem the safe setting is UNKNOWN.

A [solution]-section contains [con] and [var] sections. Each [con] and [var] section defines solution information for a single variable or constraint, specified as list of KEYWORD/value pairs, in any order, written as

```
KEYWORD=value
```

Allowed keywords are as follows:

- sk. The status of the item, where the value is one of the following strings:
  - LOW, the item is on its lower bound.
  - UPR, the item is on its upper bound.
  - FIX, it is a fixed item.
  - BAS, the item is in the basis.
  - SUPBAS, the item is super basic.
  - UNK, the status is unknown.
  - INF, the item is outside its bounds (infeasible).
- lvl Defines the level of the item.
- sl Defines the level of the dual variable associated with its lower bound.
- su Defines the level of the dual variable associated with its upper bound.
- sn Defines the level of the variable associated with its cone.
- y Defines the level of the corresponding dual variable (for constraints only).

A [var] section should always contain the items sk, lvl, sl and su. Items sl and su are not required for integer solutions.

A [con] section should always contain sk, lvl, sl, su and y.

An example of a solution section

```
[solution basic status=UNKNOWN]
[var x0] sk=LOW    lvl=5.0      [/var]
[var x1] sk=UPR    lvl=10.0     [/var]
[var x2] sk=SUPBAS lvl=2.0  sl=1.5 su=0.0 [/var]

[con c0] sk=LOW    lvl=3.0 y=0.0 [/con]
[con c0] sk=UPR    lvl=0.0 y=5.0 [/con]
[/solution]
```

- `[vendor]` This contains solver/vendor specific data. It accepts one argument, which is a vendor ID – for **MOSEK** the ID is simply `mosek` – and the section contains the subsection `parameters` defining solver parameters. When reading a vendor section, any unknown vendor can be safely ignored. This is described later.

Comments using the `#` may appear anywhere in the file. Between the `#` and the following line-break any text may be written, including markup characters.

16.3.3 Numbers

Numbers, when used for parameter values or coefficients, are written in the usual way by the `printf` function. That is, they may be prefixed by a sign (`+` or `-`) and may contain an integer part, decimal part and an exponent. The decimal point is always `.` (a dot). Some examples are

```
1
1.0
.0
1.
1e10
1e+10
1e-10
```

Some *invalid* examples are

```
e10   # invalid, must contain either integer or decimal part
.     # invalid
.e10  # invalid
```

More formally, the following standard regular expression describes numbers as used:

```
[+|-]?([0-9]+[.][0-9]*|[.][0-9]+)([eE][+|-]?[0-9]+)?
```

16.3.4 Names

Variable names, constraint names and objective name may contain arbitrary characters, which in some cases must be enclosed by quotes (single or double) that in turn must be preceded by a backslash. Unquoted names must begin with a letter (`a-z` or `A-Z`) and contain only the following characters: the letters `a-z` and `A-Z`, the digits `0-9`, braces (`{` and `}`) and underscore (`_`).

Some examples of legal names:

```
an_unquoted_name
another_name{123}
'single quoted name'
"double quoted name"
"name with \\"quote\\" in it"
"name with []s in it"
```

16.3.5 Parameters Section

In the `vendor` section solver parameters are defined inside the `parameters` subsection. Each parameter is written as

```
[p PARAMETER_NAME] value [/p]
```

where `PARAMETER_NAME` is replaced by a **MOSEK** parameter name, usually of the form `MSK_IPAR_...`, `MSK_DPAR_...` or `MSK_SPAR_...`, and the `value` is replaced by the value of that parameter; both integer values and named values may be used. Some simple examples are

```
[vendor mosek]
[parameters]
[p MSK_IPAR_OPF_MAX_TERMS_PER_LINE] 10     [/p]
[p MSK_IPAR_OPF_WRITE_PARAMETERS]   MSK_ON [/p]
[p MSK_DPAR_DATA_TOL_BOUND_INF]     1.0e18 [/p]
[/parameters]
[/vendor]
```

16.3.6 Writing OPF Files from MOSEK

To write an OPF file then make sure the file extension is .opf.

Then modify the following parameters to define what the file should contain:

iparam.opf_write_sol_bas	Include basic solution, if defined.
iparam.opf_write_sol_itg	Include integer solution, if defined.
iparam.opf_write_sol_itr	Include interior solution, if defined.
iparam.opf_write_solutions	Include solutions if they are defined. If this is off, no solutions are included.
iparam.opf_write_header	Include a small header with comments.
iparam.opf_write_problem	Include the problem itself — objective, constraints and bounds.
iparam.opf_write_parameters	Include all parameter settings.
iparam.opf_write_hints	Include hints about the size of the problem.

16.3.7 Examples

This section contains a set of small examples written in OPF and describing how to formulate linear, quadratic and conic problems.

Linear Example `lo1.opf`

Consider the example:

$$\begin{array}{lccccccccc} \text{maximize} & 3x_0 & + & 1x_1 & + & 5x_2 & + & 1x_3 & & \\ \text{subject to} & 3x_0 & + & 1x_1 & + & 2x_2 & & & = & 30, \\ & 2x_0 & + & 1x_1 & + & 3x_2 & + & 1x_3 & \geq & 15, \\ & & & 2x_1 & & & + & 3x_3 & \leq & 25, \end{array}$$

having the bounds

$$\begin{array}{ccccc} 0 & \leq & x_0 & \leq & \infty, \\ 0 & \leq & x_1 & \leq & 10, \\ 0 & \leq & x_2 & \leq & \infty, \\ 0 & \leq & x_3 & \leq & \infty. \end{array}$$

In the `OPF` format the example is displayed as shown in Listing 16.1.

Listing 16.1: Example of an OPF file for a linear problem.

```
[comment]
  The lo1 example in OPF format
[/comment]

[hints]
  [hint NUMVAR] 4 [/hint]
  [hint NUMCON] 3 [/hint]
  [hint NUMANZ] 9 [/hint]
[/hints]
```

(continues on next page)

(continued from previous page)

```
[variables disallow_new_variables]
  x1 x2 x3 x4
[/variables]

[objective maximize 'obj']
   3 x1 + x2 + 5 x3 + x4
[/objective]

[constraints]
  [con 'c1'] 3 x1 +   x2 + 2 x3         = 30 [/con]
  [con 'c2'] 2 x1 +   x2 + 3 x3 +   x4 >= 15 [/con]
  [con 'c3']        2 x2        + 3 x4 <= 25 [/con]
[/constraints]

[bounds]
  [b] 0 <= * [/b]
  [b] 0 <= x2 <= 10 [/b]
[/bounds]
```

Quadratic Example `qo1.opf`

An example of a quadratic optimization problem is

$$\begin{array}{lcl} \text{minimize} & & x_1^2 + 0.1x_2^2 + x_3^2 - x_1x_3 - x_2 \\ \text{subject to} \quad 1 & \leq & x_1 + x_2 + x_3, \\ & & x \geq 0. \end{array}$$

This can be formulated in `opf` as shown below.

Listing 16.2: Example of an OPF file for a quadratic problem.

```
[comment]
  The qo1 example in OPF format
[/comment]

[hints]
  [hint NUMVAR] 3 [/hint]
  [hint NUMCON] 1 [/hint]
  [hint NUMANZ] 3 [/hint]
  [hint NUMQNZ] 4 [/hint]
[/hints]

[variables disallow_new_variables]
  x1 x2 x3
[/variables]

[objective minimize 'obj']
  # The quadratic terms are often written with a factor of 1/2 as here,
  # but this is not required.

   - x2 + 0.5 ( 2.0 x1 ^ 2 - 2.0 x3 * x1 + 0.2 x2 ^ 2 + 2.0 x3 ^ 2 )
[/objective]

[constraints]
  [con 'c1'] 1.0 <= x1 + x2 + x3 [/con]
[/constraints]
```

(continues on next page)

(continued from previous page)

```
[bounds]
  [b] 0 <= * [/b]
[/bounds]
```

Conic Quadratic Example `cqo1.opf`

Consider the example:

$$\begin{array}{lcll} \text{minimize} & x_3 + x_4 + x_5 & & \\ \text{subject to} & x_0 + x_1 + 2x_2 & = & 1, \\ & x_0, x_1, x_2 & \geq & 0, \\ & x_3 \geq \sqrt{x_0^2 + x_1^2}, & & \\ & 2x_4 x_5 \geq x_2^2. & & \end{array}$$

Please note that the type of the cones is defined by the parameter to `[cone ...]`; the content of the `cone`-section is the names of variables that belong to the cone. The resulting OPF file is in Listing 16.3.

Listing 16.3: Example of an OPF file for a conic quadratic problem.

```
[comment]
  The cqo1 example in OPF format.
[/comment]

[hints]
  [hint NUMVAR] 6 [/hint]
  [hint NUMCON] 1 [/hint]
  [hint NUMANZ] 3 [/hint]
[/hints]

[variables disallow_new_variables]
  x1 x2 x3 x4 x5 x6
[/variables]

[objective minimize 'obj']
   x4 + x5 + x6
[/objective]

[constraints]
  [con 'c1']  x1 + x2 + 2e+00 x3 = 1e+00 [/con]
[/constraints]

[bounds]
  # We let all variables default to the positive orthant
  [b] 0 <= * [/b]

  # ...and change those that differ from the default
  [b] x4,x5,x6 free [/b]

  # Define quadratic cone: x4 >= sqrt( x1^2 + x2^2 )
  [cone quad 'k1'] x4, x1, x2 [/cone]

  # Define rotated quadratic cone: 2 x5 x6 >= x3^2
  [cone rquad 'k2'] x5, x6, x3 [/cone]
[/bounds]
```

Mixed Integer Example `milo1.opf`

Consider the mixed integer problem:

$$\begin{array}{lrcl} \text{maximize} & x_0 + 0.64x_1 & & \\ \text{subject to} & 50x_0 + 31x_1 & \leq & 250, \\ & 3x_0 - 2x_1 & \geq & -4, \\ & x_0, x_1 \geq 0 & & \text{and integer} \end{array}$$

This can be implemented in `OPF` with the file in Listing 16.4.

Listing 16.4: Example of an OPF file for a mixed-integer linear problem.

```
[comment]
  The milo1 example in OPF format
[/comment]

[hints]
  [hint NUMVAR] 2 [/hint]
  [hint NUMCON] 2 [/hint]
  [hint NUMANZ] 4 [/hint]
[/hints]

[variables disallow_new_variables]
  x1 x2
[/variables]

[objective maximize 'obj']
   x1 + 6.4e-1 x2
[/objective]

[constraints]
  [con 'c1'] 5e+1 x1 + 3.1e+1 x2 <= 2.5e+2 [/con]
  [con 'c2'] -4 <= 3 x1 - 2 x2 [/con]
[/constraints]

[bounds]
  [b] 0 <= * [/b]
[/bounds]

[integer]
  x1 x2
[/integer]
```

16.4 The CBF Format

This document constitutes the technical reference manual of the *Conic Benchmark Format* with file extension: `.cbf` or `.CBF`. It unifies linear, second-order cone (also known as conic quadratic), exponential cone, power cone and semidefinite optimization with mixed-integer variables. The format has been designed with benchmark libraries in mind, and therefore focuses on compact and easily parsable representations. The CBF format separates problem structure from the problem data.

16.4.1 How Instances Are Specified

This section defines the spectrum of conic optimization problems that can be formulated in terms of the keywords of the CBF format.

In the CBF format, conic optimization problems are considered in the following form:

$$\begin{array}{lll} \min / \max & g^{obj} & \\ & g_i \in \mathcal{K}_i, & i \in \mathcal{I}, \\ \text{s.t.} & G_i \in \mathcal{K}_i, & i \in \mathcal{I}^{PSD}, \\ & x_j \in \mathcal{K}_j, & j \in \mathcal{J}, \\ & \overline{X}_j \in \mathcal{K}_j, & j \in \mathcal{J}^{PSD}. \end{array} \tag{16.9}$$

- **Variables** are either scalar variables, x_j for $j \in \mathcal{J}$, or matrix variables, $\overline{X}_j$ for $j \in \mathcal{J}^{PSD}$. Scalar variables can also be declared as integer.
- **Constraints** are affine expressions of the variables, either scalar-valued g_i for $i \in \mathcal{I}$, or matrix-valued G_i for $i \in \mathcal{I}^{PSD}$

$$g_i = \sum_{j \in \mathcal{J}^{PSD}} \langle F_{ij}, X_j \rangle + \sum_{j \in \mathcal{J}} a_{ij} x_j + b_i,$$

$$G_i = \sum_{j \in \mathcal{J}} x_j H_{ij} + D_i.$$

- The **objective function** is a scalar-valued affine expression of the variables, either to be minimized or maximized. We refer to this expression as g^{obj}

$$g^{obj} = \sum_{j \in \mathcal{J}^{PSD}} \langle F_j^{obj}, X_j \rangle + \sum_{j \in \mathcal{J}} a_j^{obj} x_j + b^{obj}.$$

As of version 4 of the format, CBF files can represent the following non-parametric cones $\mathcal{K}$:

- **Free domain** - A cone in the linear family defined by

$$\{x \in \mathbb{R}^n\}, \text{ for } n \geq 1.$$

- **Positive orthant** - A cone in the linear family defined by

$$\{x \in \mathbb{R}^n \mid x_j \geq 0 \text{ for } j = 1, \ldots, n\}, \text{ for } n \geq 1.$$

- **Negative orthant** - A cone in the linear family defined by

$$\{x \in \mathbb{R}^n \mid x_j \leq 0 \text{ for } j = 1, \ldots, n\}, \text{ for } n \geq 1.$$

- **Fixpoint zero** - A cone in the linear family defined by

$$\{x \in \mathbb{R}^n \mid x_j = 0 \text{ for } j = 1, \ldots, n\}, \text{ for } n \geq 1.$$

- **Quadratic cone** - A cone in the second-order cone family defined by

$$\left\{ \begin{pmatrix} p \\ x \end{pmatrix} \in \mathbb{R} \times \mathbb{R}^{n-1},\ p^2 \geq x^T x,\ p \geq 0 \right\},\ \text{for } n \geq 2.$$

- **Rotated quadratic cone** - A cone in the second-order cone family defined by

$$\left\{ \begin{pmatrix} p \\ q \\ x \end{pmatrix} \in \mathbb{R} \times \mathbb{R} \times \mathbb{R}^{n-2},\ 2pq \geq x^T x,\ p \geq 0,\ q \geq 0 \right\},\ \text{for } n \geq 3.$$

- **Exponential cone** - A cone in the exponential cone family defined by

$$\mathrm{cl}(S_1) = S_1 \cup S_2$$

where,

$$S_1 = \left\{ \begin{pmatrix} t \\ s \\ r \end{pmatrix} \in \mathbb{R}^3,\ t \geq se^{\frac{r}{s}},\ s \geq 0 \right\}.$$

and,

$$S_2 = \left\{ \begin{pmatrix} t \\ s \\ r \end{pmatrix} \in \mathbb{R}^3,\ t \geq 0,\ r \leq 0,\ s = 0 \right\}.$$

- **Dual Exponential cone** - A cone in the exponential cone family defined by

$$\mathrm{cl}(S_1) = S_1 \cup S_2$$

where,

$$S_1 = \left\{ \begin{pmatrix} t \\ s \\ r \end{pmatrix} \in \mathbb{R}^3,\ et \geq (-r)e^{\frac{s}{r}},\ -r \geq 0 \right\}.$$

and,

$$S_2 = \left\{ \begin{pmatrix} t \\ s \\ r \end{pmatrix} \in \mathbb{R}^3,\ et \geq 0,\ s \geq 0,\ r = 0 \right\}.$$

- **Radial geometric mean cone** - A cone in the power cone family defined by

$$\left\{ \begin{pmatrix} p \\ x \end{pmatrix} \in \mathbb{R}^k_+ \times \mathbb{R}^1,\ \left(\prod_{j=1}^{k} p_j \right)^{\frac{1}{k}} \geq |x| \right\},\ \text{for } n = k+1 \geq 2.$$

- **Dual radial geometric mean cone** - A cone in the power cone family defined by

$$\left\{ \begin{pmatrix} p \\ x \end{pmatrix} \in \mathbb{R}^k_+ \times \mathbb{R}^1,\ \left(\prod_{j=1}^{k} kp_j \right)^{\frac{1}{k}} \geq |x| \right\},\ \text{for } n = k+1 \geq 2.$$

and, the following parametric cones:

- **Radial power cone** - A cone in the power cone family defined by

$$\left\{ \begin{pmatrix} p \\ x \end{pmatrix} \in \mathbb{R}^k_+ \times \mathbb{R}^{n-k}, \ \left(\prod_{j=1}^{k} p_j^{\alpha_j} \right)^{\frac{1}{\sigma}} \geq \|x\|_2 \right\}, \text{ for } n \geq k \geq 1.$$

  where, $\sigma = \sum_{j=1}^{k} \alpha_j$ and $\alpha = \mathbb{R}^k_{++}$.

- **Dual radial power cone** - A cone in the power cone family defined by

$$\left\{ \begin{pmatrix} p \\ x \end{pmatrix} \in \mathbb{R}^k_+ \times \mathbb{R}^{n-k}, \ \left(\prod_{j=1}^{k} \left(\frac{\sigma p_j}{\alpha_j} \right)^{\alpha_j} \right)^{\frac{1}{\sigma}} \geq \|x\|_2 \right\}, \text{ for } n \geq k \geq 1.$$

  where, $\sigma = \sum_{j=1}^{k} \alpha_j$ and $\alpha = \mathbb{R}^k_{++}$.

16.4.2 The Structure of CBF Files

This section defines how information is written in the CBF format, without being specific about the type of information being communicated.

All information items belong to exactly one of the three groups of information. These information groups, and the order they must appear in, are:

1. File format.
2. Problem structure.
3. Problem data.

The first group, file format, provides information on how to interpret the file. The second group, problem structure, provides the information needed to deduce the type and size of the problem instance. Finally, the third group, problem data, specifies the coefficients and constants of the problem instance.

Information items

The format is composed as a list of information items. The first line of an information item is the `KEYWORD`, revealing the type of information provided. The second line - of some keywords only - is the `HEADER`, typically revealing the size of information that follows. The remaining lines are the `BODY` holding the actual information to be specified.

```
KEYWORD
BODY

KEYWORD
HEADER
BODY
```

The `KEYWORD` determines how each line in the `HEADER` and `BODY` is structured. Moreover, the number of lines in the `BODY` follows either from the `KEYWORD`, the `HEADER`, or from another information item required to precede it.

File encoding and line width restrictions

The format is based on the US-ASCII printable character set with two extensions as listed below. Note, by definition, that none of these extensions can be misinterpreted as printable US-ASCII characters:

- A line feed marks the end of a line, carriage returns are ignored.
- Comment-lines may contain unicode characters in UTF-8 encoding.

The line width is restricted to 512 bytes, with 3 bytes reserved for the potential carriage return, line feed and null-terminator.

Integers and floating point numbers must follow the ISO C decimal string representation in the standard C locale. The format does not impose restrictions on the magnitude of, or number of significant digits in numeric data, but the use of 64-bit integers and 64-bit IEEE 754 floating point numbers should be sufficient to avoid loss of precision.

Comment-line and whitespace rules

The format allows single-line comments respecting the following rule:

- Lines having first byte equal to '#' (US-ASCII 35) are comments, and should be ignored. Comments are only allowed between information items.

Given that a line is not a comment-line, whitespace characters should be handled according to the following rules:

- Leading and trailing whitespace characters should be ignored.
  - The seperator between multiple pieces of information on one line, is either one or more whitespace characters.
- Lines containing only whitespace characters are empty, and should be ignored. Empty lines are only allowed between information items.

16.4.3 Problem Specification

The problem structure

The problem structure defines the objective sense, whether it is minimization and maximization. It also defines the index sets, $\mathcal{J}$, $\mathcal{J}^{PSD}$, $\mathcal{I}$ and $\mathcal{I}^{PSD}$, which are all numbered from zero, $\{0, 1, \ldots\}$, and empty until explicitly constructed.

- **Scalar variables** are constructed in vectors restricted to a conic domain, such as $(x_0, x_1) \in \mathbb{R}^2_+$, $(x_2, x_3, x_4) \in \mathcal{Q}^3$, etc. In terms of the Cartesian product, this generalizes to

$$x \in \mathcal{K}_1^{n_1} \times \mathcal{K}_2^{n_2} \times \cdots \times \mathcal{K}_k^{n_k}$$

  which in the CBF format becomes:

```
VAR
n k
K1 n1
K2 n2
...
Kk nk
```

  where $\sum_i n_i = n$ is the total number of scalar variables. The list of supported cones is found in Table 16.3. Integrality of scalar variables can be specified afterwards.

- **PSD variables** are constructed one-by-one. That is, $X_j \succeq \mathbf{0}^{n_j \times n_j}$ for $j \in \mathcal{J}^{PSD}$, constructs a matrix-valued variable of size $n_j \times n_j$ restricted to be symmetric positive semidefinite. In the CBF format, this list of constructions becomes:

```
PSDVAR
N
n1
n2
...
nN
```

where N is the total number of PSD variables.

- **Scalar constraints** are constructed in vectors restricted to a conic domain, such as $(g_0, g_1) \in \mathbb{R}^2_+$, $(g_2, g_3, g_4) \in \mathcal{Q}^3$, etc. In terms of the Cartesian product, this generalizes to

$$g \in \mathcal{K}_1^{m_1} \times \mathcal{K}_2^{m_2} \times \cdots \times \mathcal{K}_k^{m_k}$$

which in the CBF format becomes:

```
CON
m k
K1 m1
K2 m2
..
Kk mk
```

where $\sum_i m_i = m$ is the total number of scalar constraints. The list of supported cones is found in Table 16.3.

- **PSD constraints** are constructed one-by-one. That is, $G_i \succeq \mathbf{0}^{m_i \times m_i}$ for $i \in \mathcal{I}^{PSD}$, constructs a matrix-valued affine expressions of size $m_i \times m_i$ restricted to be symmetric positive semidefinite. In the CBF format, this list of constructions becomes

```
PSDCON
M
m1
m2
..
mM
```

where M is the total number of PSD constraints.

With the objective sense, variables (with integer indications) and constraints, the definitions of the many affine expressions follow in problem data.

Problem data

The problem data defines the coefficients and constants of the affine expressions of the problem instance. These are considered zero until explicitly defined, implying that instances with no keywords from this information group are, in fact, valid. Duplicating or conflicting information is a failure to comply with the standard. Consequently, two coefficients written to the same position in a matrix (or to transposed positions in a symmetric matrix) is an error.

The affine expressions of the objective, g^{obj}, of the scalar constraints, g_i, and of the PSD constraints, G_i, are defined separately. The following notation uses the standard trace inner product for matrices, $\langle X, Y \rangle = \sum_{i,j} X_{ij} Y_{ij}$.

- The affine expression of the objective is defined as

$$g^{obj} = \sum_{j \in \mathcal{J}^{PSD}} \langle F_j^{obj}, X_j \rangle + \sum_{j \in \mathcal{J}} a_j^{obj} x_j + b^{obj},$$

in terms of the symmetric matrices, F_j^{obj}, and scalars, a_j^{obj} and b^{obj}.

- The affine expressions of the scalar constraints are defined, for $i \in \mathcal{I}$, as

$$g_i = \sum_{j \in \mathcal{J}^{PSD}} \langle F_{ij}, X_j \rangle + \sum_{j \in \mathcal{J}} a_{ij} x_j + b_i,$$

  in terms of the symmetric matrices, F_{ij}, and scalars, a_{ij} and b_i.

- The affine expressions of the PSD constraints are defined, for $i \in \mathcal{I}^{PSD}$, as

$$G_i = \sum_{j \in \mathcal{J}} x_j H_{ij} + D_i,$$

  in terms of the symmetric matrices, H_{ij} and D_i.

List of cones

The format uses an explicit syntax for symmetric positive semidefinite cones as shown above. For scalar variables and constraints, constructed in vectors, the supported conic domains and their sizes are given as follows.

Table 16.3: Cones available in the CBF format

Name	CBF keyword	Cone family	Cone size
Free domain	`F`	linear	$n \geq 1$
Positive orthant	`L+`	linear	$n \geq 1$
Negative orthant	`L-`	linear	$n \geq 1$
Fixpoint zero	`L=`	linear	$n \geq 1$
Quadratic cone	`Q`	second-order	$n \geq 1$
Rotated quadratic cone	`QR`	second-order	$n \geq 2$
Exponential cone	`EXP`	exponential	$n = 3$
Dual exponential cone	`EXP*`	exponential	$n = 3$
Radial geometric mean cone	`GMEANABS`	power	$n = k + 1 \geq 2$
Dual radial geometric mean cone	`GMEANABS*`	power	$n = k + 1 \geq 2$
Radial power cone (parametric)	`POW`	power	$n \geq k \geq 1$
Dual radial power cone (parametric)	`POW*`	power	$n \geq k \geq 1$

16.4.4 File Format Keywords

VER

Description: The version of the Conic Benchmark Format used to write the file.
`HEADER`: None
`BODY`: One line formatted as:

```
INT
```

This is the version number.
Must appear exactly once in a file, as the first keyword.

POWCONES

Description: Define a lookup table for power cone domains.
`HEADER`: One line formatted as:

```
INT INT
```

This is the number of cones to be specified and the combined length of their dense parameter vectors.

`BODY`: A list of chunks each specifying the dense parameter vector of a power cone.
`CHUNKHEADER`: One line formatted as:

```
INT
```

This is the parameter vector length.

`CHUNKBODY`: A list of lines formatted as:

```
REAL
```

This is the parameter vector values. The number of lines should match the number stated in the chunk header.

The cone specified at index k (with 0-based indexing) is registered under the CBF name @k:POW.

POW*CONES

Description: Define a lookup table for dual power cone domains.
`HEADER`: One line formatted as:

```
INT INT
```

This is the number of cones to be specified and the combined length of their dense parameter vectors.

`BODY`: A list of chunks each specifying the dense parameter vector of a dual power cone.
`CHUNKHEADER`: One line formatted as:

```
INT
```

This is the parameter vector length.

`CHUNKBODY`: A list of lines formatted as:

```
REAL
```

This is the parameter vector values. The number of lines should match the number stated in the chunk header.

The cone specified at index k (with 0-based indexing) is registered under the CBF name @k:POW*.

OBJSENSE

Description: Define the objective sense.
`HEADER`: None
`BODY`: One line formatted as:

```
STR
```

having `MIN` indicates minimize, and `MAX` indicates maximize. Upper-case letters are required. Must appear exactly once in a file.

PSDVAR

Description: Construct the PSD variables.
`HEADER`: One line formatted as:

```
INT
```

This is the number of PSD variables in the problem.
`BODY`: A list of lines formatted as:

```
INT
```

This indicates the number of rows (equal to the number of columns) in the matrix-valued PSD variable. The number of lines should match the number stated in the header.

VAR

Description: Construct the scalar variables.

`HEADER`: One line formatted as:

```
INT INT
```

This is the number of scalar variables, followed by the number of conic domains they are restricted to.

`BODY`: A list of lines formatted as:

```
STR INT
```

This indicates the cone name (see Table 16.3), and the number of scalar variables restricted to this cone. These numbers should add up to the number of scalar variables stated first in the header. The number of lines should match the second number stated in the header.

INT

Description: Declare integer requirements on a selected subset of scalar variables.

`HEADER`: one line formatted as:

```
INT
```

This is the number of integer scalar variables in the problem.

`BODY`: a list of lines formatted as:

```
INT
```

This indicates the scalar variable index $j \in \mathcal{J}$. The number of lines should match the number stated in the header.

Can only be used after the keyword `VAR`.

PSDCON

Description: Construct the PSD constraints.

`HEADER`: One line formatted as:

```
INT
```

This is the number of PSD constraints in the problem.

`BODY`: A list of lines formatted as:

```
INT
```

This indicates the number of rows (equal to the number of columns) in the matrix-valued affine expression of the PSD constraint. The number of lines should match the number stated in the header.

Can only be used after these keywords: `PSDVAR`, `VAR`.

CON

Description: Construct the scalar constraints.

`HEADER`: One line formatted as:

```
INT INT
```

This is the number of scalar constraints, followed by the number of conic domains they restrict to.

`BODY`: A list of lines formatted as:

```
STR INT
```

This indicates the cone name (see Table 16.3), and the number of affine expressions restricted to this cone. These numbers should add up to the number of scalar constraints stated first in the header. The number of lines should match the second number stated in the header.

Can only be used after these keywords: `PSDVAR`, `VAR`

OBJFCOORD

Description: Input sparse coordinates (quadruplets) to define the symmetric matrices F_j^{obj}, as used in the objective.

`HEADER`: One line formatted as:

```
INT
```

This is the number of coordinates to be specified.

`BODY`: A list of lines formatted as:

```
INT INT INT REAL
```

This indicates the PSD variable index $j \in \mathcal{J}^{PSD}$, the row index, the column index and the coefficient value. The number of lines should match the number stated in the header.

OBJACOORD

Description: Input sparse coordinates (pairs) to define the scalars, a_j^{obj}, as used in the objective.

`HEADER`: One line formatted as:

```
INT
```

This is the number of coordinates to be specified.

`BODY`: A list of lines formatted as:

```
INT REAL
```

This indicates the scalar variable index $j \in \mathcal{J}$ and the coefficient value. The number of lines should match the number stated in the header.

OBJBCOORD

Description: Input the scalar, b^{obj}, as used in the objective.

`HEADER`: None.

`BODY`: One line formatted as:

```
REAL
```

This indicates the coefficient value.

FCOORD

Description: Input sparse coordinates (quintuplets) to define the symmetric matrices, F_{ij}, as used in the scalar constraints.

`HEADER`: One line formatted as:

```
INT
```

This is the number of coordinates to be specified.

`BODY`: A list of lines formatted as:

```
INT INT INT INT REAL
```

This indicates the scalar constraint index $i \in \mathcal{I}$, the PSD variable index $j \in \mathcal{J}^{PSD}$, the row index, the column index and the coefficient value. The number of lines should match the number stated in the header.

ACOORD

Description: Input sparse coordinates (triplets) to define the scalars, a_{ij}, as used in the scalar constraints.
`HEADER`: One line formatted as:

```
INT
```

This is the number of coordinates to be specified.
`BODY`: A list of lines formatted as:

```
INT INT REAL
```

This indicates the scalar constraint index $i \in \mathcal{I}$, the scalar variable index $j \in \mathcal{J}$ and the coefficient value. The number of lines should match the number stated in the header.

BCOORD

Description: Input sparse coordinates (pairs) to define the scalars, b_i, as used in the scalar constraints.
`HEADER`: One line formatted as:

```
INT
```

This is the number of coordinates to be specified.
`BODY`: A list of lines formatted as:

```
INT REAL
```

This indicates the scalar constraint index $i \in \mathcal{I}$ and the coefficient value. The number of lines should match the number stated in the header.

HCOORD

Description: Input sparse coordinates (quintuplets) to define the symmetric matrices, H_{ij}, as used in the PSD constraints.
`HEADER`: One line formatted as:

```
INT
```

This is the number of coordinates to be specified.
`BODY`: A list of lines formatted as

```
INT INT INT INT REAL
```

This indicates the PSD constraint index $i \in \mathcal{I}^{PSD}$, the scalar variable index $j \in \mathcal{J}$, the row index, the column index and the coefficient value. The number of lines should match the number stated in the header.

DCOORD

Description: Input sparse coordinates (quadruplets) to define the symmetric matrices, D_i, as used in the PSD constraints.
`HEADER`: One line formatted as

```
INT
```

This is the number of coordinates to be specified.
`BODY`: A list of lines formatted as:

```
INT INT INT REAL
```

This indicates the PSD constraint index $i \in \mathcal{I}^{PSD}$, the row index, the column index and the coefficient value. The number of lines should match the number stated in the header.

16.4.5 CBF Format Examples

Minimal Working Example

The conic optimization problem (16.10) , has three variables in a quadratic cone - first one is integer - and an affine expression in domain 0 (equality constraint).

$$\begin{array}{ll} \text{minimize} & 5.1\,x_0 \\ \text{subject to} & 6.2\,x_1 + 7.3\,x_2 - 8.4 \in \{0\} \\ & x \in \mathcal{Q}^3,\ x_0 \in \mathbb{Z}. \end{array} \tag{16.10}$$

Its formulation in the Conic Benchmark Format begins with the version of the CBF format used, to safeguard against later revisions.

```
VER
4
```

Next follows the problem structure, consisting of the objective sense, the number and domain of variables, the indices of integer variables, and the number and domain of scalar-valued affine expressions (i.e., the equality constraint).

```
OBJSENSE
MIN

VAR
3 1
Q 3

INT
1
0

CON
1 1
L= 1
```

Finally follows the problem data, consisting of the coefficients of the objective, the coefficients of the constraints, and the constant terms of the constraints. All data is specified on a sparse coordinate form.

```
OBJACOORD
1
0 5.1

ACOORD
2
0 1 6.2
0 2 7.3

BCOORD
1
0 -8.4
```

This concludes the example.

Mixing Linear, Second-order and Semidefinite Cones

The conic optimization problem (16.11), has a semidefinite cone, a quadratic cone over unordered subindices, and two equality constraints.

$$\begin{array}{lll} \text{minimize} & \left\langle \begin{bmatrix} 2 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 2 \end{bmatrix}, X_1 \right\rangle + x_1 & \\ \text{subject to} & \left\langle \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, X_1 \right\rangle + x_1 & = \quad 1.0\,, \\ & \left\langle \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, X_1 \right\rangle + x_0 + x_2 & = \quad 0.5\,, \\ & x_1 \geq \sqrt{x_0^2 + x_2^2}\,, & \\ & X_1 \succeq \mathbf{0}\,. & \end{array} \tag{16.11}$$

The equality constraints are easily rewritten to the conic form, $(g_0, g_1) \in \{0\}^2$, by moving constants such that the right-hand-side becomes zero. The quadratic cone does not fit under the `VAR` keyword in this variable permutation. Instead, it takes a scalar constraint $(g_2, g_3, g_4) = (x_1, x_0, x_2) \in \mathcal{Q}^3$, with scalar variables constructed as $(x_0, x_1, x_2) \in \mathbb{R}^3$. Its formulation in the CBF format is reported in the following list

```
# File written using this version of the Conic Benchmark Format:
#     | Version 4.
VER
4

# The sense of the objective is:
#     | Minimize.
OBJSENSE
MIN

# One PSD variable of this size:
#     | Three times three.
PSDVAR
1
3

# Three scalar variables in this one conic domain:
#     | Three are free.
VAR
3 1
F 3

# Five scalar constraints with affine expressions in two conic domains:
#     | Two are fixed to zero.
#     | Three are in conic quadratic domain.
CON
5 2
L= 2
Q 3

# Five coordinates in F^{obj}_j coefficients:
#     | F^{obj}[0][0,0] = 2.0
#     | F^{obj}[0][1,0] = 1.0
#     | and more...
OBJFCOORD
5
```

(continues on next page)

(continued from previous page)

```
0 0 0 2.0
0 1 0 1.0
0 1 1 2.0
0 2 1 1.0
0 2 2 2.0

# One coordinate in a^{obj}_j coefficients:
#     | a^{obj}[1] = 1.0
OBJACOORD
1
1 1.0

# Nine coordinates in F_ij coefficients:
#     | F[0,0][0,0] = 1.0
#     | F[0,0][1,1] = 1.0
#     | and more...
FCOORD
9
0 0 0 0 1.0
0 0 1 1 1.0
0 0 2 2 1.0
1 0 0 0 1.0
1 0 1 0 1.0
1 0 2 0 1.0
1 0 1 1 1.0
1 0 2 1 1.0
1 0 2 2 1.0

# Six coordinates in a_ij coefficients:
#     | a[0,1] = 1.0
#     | a[1,0] = 1.0
#     | and more...
ACOORD
6
0 1 1.0
1 0 1.0
1 2 1.0
2 1 1.0
3 0 1.0
4 2 1.0

# Two coordinates in b_i coefficients:
#     | b[0] = -1.0
#     | b[1] = -0.5
BCOORD
2
0 -1.0
1 -0.5
```

Mixing Semidefinite Variables and Linear Matrix Inequalities

The standard forms in semidefinite optimization are usually based either on semidefinite variables or linear matrix inequalities. In the CBF format, both forms are supported and can even be mixed as shown.

$$\begin{array}{lll} \text{minimize} & \left\langle \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, X_1 \right\rangle + x_1 + x_2 + 1 & \\ \text{subject to} & \left\langle \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, X_1 \right\rangle - x_1 - x_2 & \geq \quad 0.0\,, \\ & x_1 \begin{bmatrix} 0 & 1 \\ 1 & 3 \end{bmatrix} + x_2 \begin{bmatrix} 3 & 1 \\ 1 & 0 \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \succeq \quad \mathbf{0}\,, \\ & X_1 \succeq \mathbf{0}\,. & \end{array} \tag{16.12}$$

Its formulation in the CBF format is written in what follows

```
# File written using this version of the Conic Benchmark Format:
#     | Version 4.
VER
4

# The sense of the objective is:
#     | Minimize.
OBJSENSE
MIN

# One PSD variable of this size:
#     | Two times two.
PSDVAR
1
2

# Two scalar variables in this one conic domain:
#     | Two are free.
VAR
2 1
F 2

# One PSD constraint of this size:
#     | Two times two.
PSDCON
1
2

# One scalar constraint with an affine expression in this one conic domain:
#     | One is greater than or equal to zero.
CON
1 1
L+ 1

# Two coordinates in F^{obj}_j coefficients:
#     | F^{obj}[0][0,0] = 1.0
#     | F^{obj}[0][1,1] = 1.0
OBJFCOORD
2
0 0 0 1.0
0 1 1 1.0

# Two coordinates in a^{obj}_j coefficients:
```

(continues on next page)

(continued from previous page)

```
#     | a^{obj}[0] = 1.0
#     | a^{obj}[1] = 1.0
OBJACOORD
2
0 1.0
1 1.0

# One coordinate in b^{obj} coefficient:
#     | b^{obj} = 1.0
OBJBCOORD
1.0

# One coordinate in F_ij coefficients:
#     | F[0,0][1,0] = 1.0
FCOORD
1
0 0 1 0 1.0

# Two coordinates in a_ij coefficients:
#     | a[0,0] = -1.0
#     | a[0,1] = -1.0
ACOORD
2
0 0 -1.0
0 1 -1.0

# Four coordinates in H_ij coefficients:
#     | H[0,0][1,0] = 1.0
#     | H[0,0][1,1] = 3.0
#     | and more...
HCOORD
4
0 0 1 0 1.0
0 0 1 1 3.0
0 1 0 0 3.0
0 1 1 0 1.0

# Two coordinates in D_i coefficients:
#     | D[0][0,0] = -1.0
#     | D[0][1,1] = -1.0
DCOORD
2
0 0 0 -1.0
0 1 1 -1.0
```

The exponential cone

The conic optimization problem (16.13), has one equality constraint, one quadratic cone constraint and an exponential cone constraint.

$$\begin{array}{ll} \text{minimize} & x_0 - x_3 \\ \text{subject to} & x_0 + 2\,x_1 - x_2 \in \{0\} \\ & (5.0, x_0, x_1) \in \mathcal{Q}^3 \\ & (x_2, 1.0, x_3) \in EXP. \end{array} \tag{16.13}$$

The nonlinear conic constraints enforce $\sqrt{x_0^2 + x_1^2} \leq 0.5$ and $x_3 \leq \log(x_2)$.

```
# File written using this version of the Conic Benchmark Format:
#      | Version 3.
VER
3

# The sense of the objective is:
#      | Minimize.
OBJSENSE
MIN

# Four scalar variables in this one conic domain:
#      | Four are free.
VAR
4 1
F 4

# Seven scalar constraints with affine expressions in three conic domains:
#      | One is fixed to zero.
#      | Three are in conic quadratic domain.
#      | Three are in exponential cone domain.
CON
7 3
L= 1
Q 3
EXP 3

# Two coordinates in a^{obj}_j coefficients:
#      | a^{obj}[0] = 1.0
#      | a^{obj}[3] = -1.0
OBJACOORD
2
0 1.0
3 -1.0

# Seven coordinates in a_ij coefficients:
#      | a[0,0] = 1.0
#      | a[0,1] = 2.0
#      | and more...
ACOORD
7
0 0 1.0
0 1 2.0
0 2 -1.0
2 0 1.0
3 1 1.0
4 2 1.0
6 3 1.0
```

(continues on next page)

(continued from previous page)

```
# Two coordinates in b_i coefficients:
#      | b[1] = 5.0
#      | b[5] = 1.0
BCOORD
2
1 5.0
5 1.0
```

Parametric cones

The problem (16.14), has three variables in a power cone with parameter $\alpha_1 = (1, 1)$ and two power cone constraints each with parameter $\alpha_0 = (8, 1)$.

$$\begin{array}{ll} \text{minimize} & x_3 \\ \text{subject to} & (1.0, x_1, x_1 + x_2) \in POW_{\alpha_0} \\ & (1.0, x_2, x_1 + x_2) \in POW_{\alpha_0} \\ & x \in POW_{\alpha_1}. \end{array} \tag{16.14}$$

The nonlinear conic constraints enforce $x_3 \leq x_1 x_2$ and $x_1 + x_2 \leq \min(x_1^{\frac{1}{9}}, x_2^{\frac{1}{9}})$.

```
# File written using this version of the Conic Benchmark Format:
#      | Version 3.
VER
3

# Two power cone domains defined in a total of four parameters:
#      | @0:POW (specification 0) has two parameters:
#      | alpha[0] = 8.0.
#      | alpha[1] = 1.0.
#      | @1:POW (specification 1) has two parameters:
#      | alpha[0] = 1.0.
#      | alpha[1] = 1.0.
POWCONES
2 4
2
8.0
1.0
2
1.0
1.0

# The sense of the objective is:
#      | Maximize.
OBJSENSE
MAX

# Three scalar variable in this one conic domain:
#      | Three are in power cone domain (specification 1).
VAR
3 1
@1:POW 3

# Six scalar constraints with affine expressions in two conic domains:
#      | Three are in power cone domain (specification 0).
#      | Three are in power cone domain (specification 0).
```

(continues on next page)

(continued from previous page)

```
CON
6 2
@0:POW 3
@0:POW 3

# One coordinate in a^{obj}_j coefficients:
#      | a^{obj}[2] = 1.0
OBJACOORD
1
2 1.0

# Six coordinates in a_ij coefficients:
#      | a[1,0] = 1.0
#      | a[2,0] = 1.0
#      | and more...
ACOORD
6
1 0 1.0
2 0 1.0
2 1 1.0
4 1 1.0
5 0 1.0
5 1 1.0

# Two coordinates in b_i coefficients:
#      | b[0] = 1.0
#      | b[3] = 1.0
BCOORD
2
0 1.0
3 1.0
```

16.5 The PTF Format

The PTF format is a human-readable, natural text format that supports all linear, conic and mixed-integer features.

16.5.1 The overall format

The format is indentation based, where each section is started by a head line and followed by a section body with deeper indentation that the head line. For example:

```
Header line
   Body line 1
   Body line 1
   Body line 1
```

Section can also be nested:

```
Header line A
   Body line in A
   Header line A.1
      Body line in A.1
      Body line in A.1
   Body line in A
```

The indentation of blank lines is ignored, so a subsection can contain a blank line with no indentation. The character # defines a line comment and anything between the # character and the end of the line is ignored.

In a PTF file, the first section must be a `Task` section. The order of the remaining section is arbitrary, and sections may occur multiple times or not at all.

MOSEK will ignore any top-level section it does not recognize.

Names

In the description of the format we use following definitions for name strings:

```
NAME: PLAIN_NAME | QUOTED_NAME
PLAIN_NAME: [a-zA-Z_] [a-zA-Z0-9_-.!|]
QUOTED_NAME: "'" ( [^'\\\r\n] | "\\" ( [\\rn] | "x" [0-9a-fA-F] [0-9a-fA-F] ) )* "'"
```

Expressions

An expression is a sum of terms. A term is either a linear term (a coefficient and a variable name, where the coefficient can be left out if it is 1.0), or a matrix inner product.

An expression:

```
EXPR: EMPTY | ( [+-] TERM )*
TERM: LINEAR_TERM | MATRIX_TERM
```

A linear term

```
LINEAR_TERM: FLOAT? NAME
```

A matrix term

```
MATRIX_TERM: "<" ( [+-] FLOAT? NAME)* ";" NAME ">"
```

Here the right-hand name is the name of a (semidefinite) matrix variable, and the left-hand side is a sum of symmetric matrices. The actual matrices are defined in a separate section.

Expressions can span multiple lines by giving subsequent lines a deeper indentation.

For example following two section are equivalent:

```
# Everything on one line:
+ x1 + x2 + x3 + x4

# Split into multiple lines:
+ x1
  + x2
  + x3
  + x4
```

16.5.2 `Task` section

The first section of the file must be a `Task`. The text in this section is not used and may contain comments, or meta-information from the writer or about the content.

Format:

```
Task NAME
   Anything goes here...
```

`NAME` is a the task name.

16.5.3 `Objective` section

The `Objective` section defines the objective name, sense and function. The format:

```
"Objective" NAME?
   ( "Minimize" | "Maximize" ) EXPR
```

For example:

```
Objective 'obj'
   Minimize + x1 + 0.2 x2 + < M1 ; X1 >
```

16.5.4 `Constraints` section

The constraints section defines a series of constraints. A constraint defines a term $A \cdot x + b \in K$. For linear constraints `A` is just one row, while for conic constraints it can be multiple rows. If a constraint spans multiple rows these can either be written inline separated by semi-colons, or each expression in a separete sub-section.

Simple linear constraints:

```
"Constraints"
  NAME? "[" [-+] (FLOAT | "inf") (";" [-+] (FLOAT | "inf") )? "]" EXPR
```

If the brackets contain two values, they are used as upper and lower bounds. It they contain one value the constraint is an equality.

For example:

```
Constraints
  # Ranged constraint
  'c1' [0;10] + x1 + x2 + x3
  # Fixed constraint, expression equals to 0
  [0] + x1 + x2 + x3
  # Nonnegative constraint
  [0;+inf] + x1 + x2 + x3
```

Constraint blocks put the expression either in a subsection or inline. The cone type (domain) is written in the brackets, and **MOSEK** currently supports following types:

- **Major (primal) cones:**
  - `QUAD(N)` or `SOC(N)`: Second order cone of dimension `N`.
  - `RQUAD(N)` or `RSOC(N)`: Rotated second order cone of dimension `N`.
  - `PEXP`: Primal exponential cone of dimension 3.
  - `PPOW(N,P)`: Primal power cone of dimension `N` with parameter `P` (float between 0 and 1).
  - `PPOW(N;ALPHA)`: Primal power cone of dimension `N` with exponent sequence `ALPHA` (comma-separated list of floats).
  - `PGEOMEAN(N)`: Primal geometric mean cone of domension `N`.
  - `SVECPSD(N)`: Vectorized symmetric positive semidefinite cone of dimension `N` (`N` must be of the form `D*(D+1)/2`).
- **Dual cones:**
  - `DEXP`: Dual exponential cone of dimension 3.
  - `DPOW(N,P)`: Dual power cone of dimension `N` with parameter `P` (float between 0 and 1).
  - `DPOW(N;ALPHA)`: Dual power cone of dimension `N` with exponent sequence `ALPHA` (comma-separated list of floats).
  - `DGEOMEAN(N)`: Dual geometric mean cone of domension `N`.
- **Linear cones:**
  - `FREE(N)` The free (unbounded) cone of dimension `N`.
  - `POSITIVE(N)` The non-negative cone of dimension `N`.

- `NEGATIVE(N)` The non-positive cone of dimension `N`.
- `ZERO(N)` The zero-cone of dimension `N`.

See Sec. 15.11 for definitions of the parameters.

```
"Constraints"
  NAME? "[" DOMAIN "]" EXPR_LIST
```

For example:

```
Constraints
   'K1' [PPOW(5;3,1)]
      + x1 + x2
      + x2 + x3
      + 1.0
      + x1
      + x3
   'K2' [RQUAD(3)]
      + x1 + x2
      + x2 + x3
      + x3 + x1
```

16.5.5 `Variables` section

Any variable used in an expression must be defined in a variable section. The variable section defines each variable domain.

```
"Variables"
   NAME "[" [-+] (FLOAT | "inf") (";" [-+] (FLOAT | "inf") )? "]"
   NAME "[" "PSD" (INT) "]"
```

For example, a linear variable

```
Variables
   # Nonnegative variable
   x1 [0;inf]
   # Ranged variable
   x2 [0;1]
   # Fixed variable
   x3 [5.0]
   # 5-dimensional symmetric matrix variable
   X [PSD(5)]
```

16.5.6 `Integer` section

This section contains a list of variables that are integral. For example:

```
Integer
   x1 x2 x3
```

16.5.7 `SymmetricMatrixes` section

This section defines the symmetric matrixes used for matrix coefficients in matrix inner product terms. The section lists named matrixes, each with a size and a number of non-zeros. Only non-zeros in the lower triangular part should be defined.

```
"SymmetricMatrixes"
   NAME "SYMMAT" "(" INT ")"  ( "(" INT "," INT "," FLOAT ")" )*
   ...
```

For example:

```
SymmetricMatrixes
   M1 SYMMAT(3) (0,0,1.0) (1,1,2.0) (2,1,0.5)
   M2 SYMMAT(3)
      (0,0,1.0)
      (1,1,2.0)
      (2,1,0.5)
```

16.5.8 `Solutions` section

Each subsection defines a solution. A solution defines for each constraint and for each variable exactly one primal value and either one (for conic domains) or two (for linear domains) dual values. The values follow the same logic as in the **MOSEK** C API. A primal and a dual solution status defines the meaning of the values primal and dual (solution, certificate, unknown, etc.)

The format is this:

```
"Solutions"
   "Solution" WHICHSOL
      "ProblemStatus" PROSTA PROSTA?
   "SolutionStatus" SOLSTA SOLSTA?
   "Objective" FLOAT FLOAT_OR_NONE
   "Variables"
      # Linear variable status: level, slx, sux
      NAME "[" STATUS "]" FLOAT FLOAT_OR_NONE FLOAT_OR_NONE
   "Constraints"
      # Linear variable status: level, slx, sux
      NAME "[" STATUS "]" FLOAT FLOAT_OR_NONE FLOAT_OR_NONE
      # Conic constraint status: level, doty
      NAME
         "[" STATUS "]" FLOAT FLOAT_OR_NONE
```

Nonexistent values (for example, dual values for an integer solution) are replaced with a single dot (`.`):

```
FLOAT_OR_NONE = FLOAT | .
```

Following values for `WHICHSOL` are supported:

- `interior` Interior solution, the result of an interior-point solver.
- `basic` Basic solution, as produced by a simplex solver.
- `integer` Integer solution, the solution to a mixed-integer problem. This does not define a dual solution.

Following values for `PROSTA` are supported:

- `unknown` The problem status is unknown
- `feasible` The problem has been proven feasible
- `infeasible` The problem has been proven infeasible
- `illposed` The problem has been proved to be ill posed
- `infeasible_or_unbounded` The problem is infeasible or unbounded

Following values for `SOLSTA` are supported:

- `unknown` The solution status is unknown
- `feasible` The solution is feasible
- `optimal` The solution is optimal
- `infeas_cert` The solution is a certificate of infeasibility
- `illposed_cert` The solution is a certificate of illposedness

Following values for `STATUS` are supported:

- `unknown` The value is unknown
- `super_basic` The value is super basic
- `at_lower` The value is basic and at its lower bound
- `at_upper` The value is basic and at its upper bound
- `fixed` The value is basic fixed
- `infinite` The value is at infinity

16.5.9 Examples

Linear example `lo1.ptf`

```
Task ''
    # Written by MOSEK v10.0.13
    # problemtype: Linear Problem
    # number of linear variables: 4
    # number of linear constraints: 3
    # number of old-style A nonzeros: 9
Objective obj
    Maximize + 3 x1 + x2 + 5 x3 + x4
Constraints
    c1 [3e+1] + 3 x1 + x2 + 2 x3
    c2 [1.5e+1;+inf] + 2 x1 + x2 + 3 x3 + x4
    c3 [-inf;2.5e+1] + 2 x2 + 3 x4
Variables
    x1 [0;+inf]
    x2 [0;1e+1]
    x3 [0;+inf]
    x4 [0;+inf]
```

Conic quadratic example `cqo1.ptf`

```
Task ''
    # Written by MOSEK v10.0.17
    # problemtype: Conic Problem
    # number of linear variables: 6
    # number of linear constraints: 1
    # number of  old-style cones: 0
    # number of positive semidefinite variables: 0
    # number of positive semidefinite matrixes: 0
    # number of affine conic constraints: 2
    # number of disjunctive constraints: 0
    # number scalar affine expressions/nonzeros : 6/6
    # number of old-style A nonzeros: 3
Objective obj
    Minimize + x4 + x5 + x6
Constraints
    c1 [1] + x1 + x2 + 2 x3
    k1 [QUAD(3)]
        @ac1: + x4
        @ac2: + x1
        @ac3: + x2
    k2 [RQUAD(3)]
        @ac4: + x5
        @ac5: + x6
        @ac6: + x3
Variables
    x4
    x1 [0;+inf]
    x2 [0;+inf]
    x5
    x6
    x3 [0;+inf]
```

Power cone example `cqo1.ptf`

```
Task ''
Objective ''
    Maximize - x0 + x3 + x4
Constraints
    c0 [2] + x0 + x1 + 5e-1 x2
    C1 [PPOW(3,2e-1)]
        + x0
        + x1
        + x3
    C2 [PPOW(3;4.0,6.0)]
        + x2
        + x5
        + x4
Variables
    x0
    x1
    x2
    x3
    x4
    x5 [1.0]
```

Disjunctive example djc1.ptf

```
Task djc1
Objective ''
    Minimize + 2 'x[0]' + 'x[1]' + 3 'x[2]' + 'x[3]'
Constraints
    @c0 [-10;+inf] + 'x[0]' + 'x[1]' + 'x[2]' + 'x[3]'
    @D0 [OR]
        [AND]
            [NEGATIVE(1)]
                 + 'x[0]' - 2 'x[1]' + 1
            [ZERO(2)]
                 + 'x[2]'
                 + 'x[3]'
        [AND]
            [NEGATIVE(1)]
                 + 'x[2]' - 3 'x[3]' + 2
            [ZERO(2)]
                 + 'x[0]'
                 + 'x[1]'
    @D1 [OR]
        [ZERO(1)]
             + 'x[0]' - 2.5
        [ZERO(1)]
             + 'x[1]' - 2.5
        [ZERO(1)]
             + 'x[2]' - 2.5
        [ZERO(1)]
             + 'x[3]' - 2.5
Variables
    'x[0]'
    'x[1]'
    'x[2]'
    'x[3]'
```

Semidefinite example sdo1.ptf

```
Task ''
    # Written by MOSEK v10.0.17
    # problemtype: Conic Problem
    # number of linear variables: 3
    # number of linear constraints: 0
    # number of  old-style cones: 0
    # number of positive semidefinite variables: 1
    # number of positive semidefinite matrixes: 3
    # number of affine conic constraints: 2
    # number of disjunctive constraints: 0
    # number scalar affine expressions/nonzeros : 5/6
    # number of old-style A nonzeros: 0
Objective ''
    Minimize + @x0 + <M0;@X0>
Constraints
    @C0 [ZERO(2)]
        @ac0: + @x0 + < + M1;@X0> - 1
        @ac1: + @x1 + @x2 + < + M2;@X0> - 0.5
    @C1 [QUAD(3)]
```

(continues on next page)

(continued from previous page)

```
        @ac2: + @x0
        @ac3: + @x1
        @ac4: + @x2
Variables
    @x0
    @x1
    @x2
    @X0 [PSD(3)]
SymmetricMatrixes
    M0 SYMMAT(3) (0,0,2) (1,0,1) (1,1,2) (2,1,1) (2,2,2)
    M1 SYMMAT(3) (0,0,1) (1,1,1) (2,2,1)
    M2 SYMMAT(3) (0,0,1) (1,0,1) (1,1,1) (2,0,1) (2,1,1) (2,2,1)
```

16.6 The Task Format

The Task format is **MOSEK**'s native binary format. It contains a complete image of a **MOSEK** task, i.e.

- Problem data: Linear, conic, semidefinite and quadratic data
- Problem item names: Variable names, constraints names, cone names etc.
- Parameter settings
- Solutions

There are a few things to be aware of:

- Status of a solution read from a file will *always* be unknown.
- Parameter settings in a task file *always override* any parameters set on the command line or in a parameter file.

The format is based on the *TAR* (USTar) file format. This means that the individual pieces of data in a `.task` file can be examined by unpacking it as a *TAR* file. Please note that the inverse may not work: Creating a file using *TAR* will most probably not create a valid **MOSEK** Task file since the order of the entries is important.

16.7 The JSON Format

MOSEK provides the possibility to read/write problems and solutions in JSON format. The official JSON website http://www.json.org provides plenty of information along with the format definition. JSON is an industry standard for data exchange and JSON files can be easily written and read in most programming languages using dedicated libraries.

MOSEK uses two JSON-based formats:

- **JTASK**, for storing problem instances together with solutions and parameters. The JTASK format contains the same information as a native **MOSEK** task *task format*, that is a very close representation of the internal data storage in the task object.

  You can write a JTASK file specifying the extension `.jtask`. When the parameter *iparam.write_json_indentation* is set the JTASK file will be indented to slightly improve readability.

- **JSOL**, for storing solutions and information items.

  You can write a JSOL solution file using *Task.writejsonsol*. When the parameter *iparam.write_json_indentation* is set the JSOL file will be indented to slightly improve readability.

  You can read a JSOL solution into an existing task file using *Task.readjsonsol*. Only the `Task/solutions` section of the data will be taken into consideration.

16.7.1 JTASK Specification

The JTASK is a dictionary containing the following sections. All sections are optional and can be omitted if irrelevant for the problem.

- `$schema`: JSON schema.
- `Task/name`: The name of the task (string).
- `Task/INFO`: Information about problem data dimensions and similar. These are treated as hints when reading the file.
  - `numvar`: number of variables (int32).
  - `numcon`: number of constraints (int32).
  - `numcone`: number of cones (int32, deprecated).
  - `numbarvar`: number of symmetric matrix variables (int32).
  - `numanz`: number of nonzeros in A (int64).
  - `numsymmat`: number of matrices in the symmetric matrix storage E (int64).
  - `numafe`: number of affine expressions in AFE storage (int64).
  - `numfnz`: number of nonzeros in F (int64).
  - `numacc`: number of affine conic constraints (ACCs) (int64).
  - `numdjc`: number of disjunctive constraints (DJCs) (int64).
  - `numdom`: number of domains (int64).
  - `mosekver`: MOSEK version (list(int32)).
- `Task/data`: Numerical and structural data of the problem.
  - `var`: Information about variables. All fields present must have the same length as `bk`. All or none of `bk`, `bl`, and `bu` must appear.
    - `name`: Variable names (list(string)).
    - `bk`: Bound keys (list(string)).
    - `bl`: Lower bounds (list(double)).
    - `bu`: Upper bounds (list(double)).
    - `type`: Variable types (list(string)).
  - `con`: Information about linear constraints. All fields present must have the same length as `bk`. All or none of `bk`, `bl`, and `bu` must appear.
    - `name`: Constraint names (list(string)).
    - `bk`: Bound keys (list(string)).
    - `bl`: Lower bounds (list(double)).
    - `bu`: Upper bounds (list(double)).
  - `barvar`: Information about symmetric matrix variables. All fields present must have the same length as `dim`.
    - `name`: Barvar names (list(string)).
    - `dim`: Dimensions (list(int32)).
  - `objective`: Information about the objective.
    - `name`: Objective name (string).
    - `sense`: Objective sense (string).
    - `c`: The linear part c of the objective as a sparse vector. Both arrays must have the same length.
      - `subj`: indices of nonzeros (list(int32)).
      - `val`: values of nonzeros (list(double)).
    - `cfix`: Constant term in the objective (double).

    * `Q`: The quadratic part Q^o of the objective as a sparse matrix, only lower-triangular part included. All arrays must have the same length.
      · `subi`: row indices of nonzeros (list(int32)).
      · `subj`: column indices of nonzeros (list(int32)).
      · `val`: values of nonzeros (list(double)).
    * `barc`: The semidefinite part $\overline{C}$ of the objective (list). Each element of the list is a list describing one entry $\overline{C}_j$ using three fields:
      · index j (int32).
      · weights of the matrices from the storage E forming $\overline{C}_j$ (list(double)).
      · indices of the matrices from the storage E forming $\overline{C}_j$ (list(int64)).
- `A`: The linear constraint matrix A as a sparse matrix. All arrays must have the same length.
    * `subi`: row indices of nonzeros (list(int32)).
    * `subj`: column indices of nonzeros (list(int32)).
    * `val`: values of nonzeros (list(double)).
- `bara`: The semidefinite part $\overline{A}$ of the constraints (list). Each element of the list is a list describing one entry $\overline{A}_{ij}$ using four fields:
    * index i (int32).
    * index j (int32).
    * weights of the matrices from the storage E forming $\overline{A}_{ij}$ (list(double)).
    * indices of the matrices from the storage E forming $\overline{A}_{ij}$ (list(int64)).
- `AFE`: The affine expression storage.
    * `numafe`: number of rows in the storage (int64).
    * `F`: The matrix F as a sparse matrix. All arrays must have the same length.
      · `subi`: row indices of nonzeros (list(int64)).
      · `subj`: column indices of nonzeros (list(int32)).
      · `val`: values of nonzeros (list(double)).
    * `g`: The vector g of constant terms as a sparse vector. Both arrays must have the same length.
      · `subi`: indices of nonzeros (list(int64)).
      · `val`: values of nonzeros (list(double)).
    * `barf`: The semidefinite part $\overline{F}$ of the expressions in AFE storage (list). Each element of the list is a list describing one entry $\overline{F}_{ij}$ using four fields:
      · index i (int64).
      · index j (int32).
      · weights of the matrices from the storage E forming $\overline{F}_{ij}$ (list(double)).
      · indices of the matrices from the storage E forming $\overline{F}_{ij}$ (list(int64)).
- `domains`: Information about domains. All fields present must have the same length as `type`.
    * `name`: Domain names (list(string)).
    * `type`: Description of the type of each domain (list). Each element of the list is a list describing one domain using at least one field:
      · domain type (string).
      · (except `pexp`, `dexp`) dimension (int64).
      · (only `ppow`, `dpow`) weights (list(double)).
- `ACC`: Information about affine conic constraints (ACC). All fields present must have the same length as `domain`.
    * `name`: ACC names (list(string)).
    * `domain`: Domains (list(int64)).
    * `afeidx`: AFE indices, grouped by ACC (list(list(int64))).
    * `b`: constant vectors b, grouped by ACC (list(list(double))).

- `DJC`: Information about disjunctive constraints (DJC). All fields present must have the same length as `termsize`.
  - `name`: DJC names (list(string)).
  - `termsize`: Term sizes, grouped by DJC (list(list(int64))).
  - `domain`: Domains, grouped by DJC (list(list(int64))).
  - `afeidx`: AFE indices, grouped by DJC (list(list(int64))).
  - `b`: constant vectors b, grouped by DJC (list(list(double))).
- `MatrixStore`: The symmetric matrix storage E (list). Each element of the list is a list describing one entry E using four fields in sparse matrix format, lower-triangular part only:
  - dimension (int32).
  - row indices of nonzeros (list(int32)).
  - column indices of nonzeros (list(int32)).
  - values of nonzeros (list(double)).
- `Q`: The quadratic part Q^c of the constraints (list). Each element of the list is a list describing one entry Q_i^c using four fields in sparse matrix format, lower-triangular part only:
  - the row index i (int32).
  - row indices of nonzeros (list(int32)).
  - column indices of nonzeros (list(int32)).
  - values of nonzeros (list(double)).
- `qcone` (deprecated). The description of cones. All fields present must have the same length as `type`.
  - `name`: Cone names (list(string)).
  - `type`: Cone types (list(string)).
  - `par`: Additional cone parameters (list(double)).
  - `members`: Members, grouped by cone (list(list(int32))).

- `Task/solutions`: Solutions. This section can contain up to three subsections called:
  - `interior`
  - `basic`
  - `integer`

  corresponding to the three solution types in MOSEK. Each of these sections has the same structure:
  - `prosta`: problem status (string).
  - `solsta`: solution status (string).
  - `xx`, `xc`, `y`, `slc`, `suc`, `slx`, `sux`, `snx`: one for each component of the solution of the same name (list(double)).
  - `skx`, `skc`, `skn`: status keys (list(string)).
  - `doty`: the dual $\dot{y}$ solution, grouped by ACC (list(list(double))).
  - `barx`, `bars`: the primal/dual semidefinite solution, grouped by matrix variable (list(list(double))).

- `Task/parameters`: Parameters.
  - `iparam`: Integer parameters (dictionary). A dictionary with entries of the form `name:value`, where `name` is a shortened parameter name (without leading `MSK_IPAR_`) and `value` is either an integer or string if the parameter takes values from an enum.
  - `dparam`: Double parameters (dictionary). A dictionary with entries of the form `name:value`, where `name` is a shortened parameter name (without leading `MSK_DPAR_`) and `value` is a double.
  - `sparam`: String parameters (dictionary). A dictionary with entries of the form `name:value`, where `name` is a shortened parameter name (without leading `MSK_SPAR_`) and `value` is a string. Note that this section is allowed but MOSEK ignores it both when writing and reading JTASK files.

16.7.2 JSOL Specification

The JSOL is a dictionary containing the following sections. All sections are optional and can be omitted if irrelevant for the problem.

- `$schema`: JSON schema.
- `Task/name`: The name of the task (string).
- `Task/solutions`: Solutions. This section can contain up to three subsections called:
  - `interior`
  - `basic`
  - `integer`

  corresponding to the three solution types in MOSEK. Each of these section has the same structure:
  - `prosta`: problem status (string).
  - `solsta`: solution status (string).
  - `xx`, `xc`, `y`, `slc`, `suc`, `slx`, `sux`, `snx`: one for each component of the solution of the same name (list(double)).
  - `skx`, `skc`, `skn`: status keys (list(string)).
  - `doty`: the dual $\dot{y}$ solution, grouped by ACC (list(list(double))).
  - `barx`, `bars`: the primal/dual semidefinite solution, grouped by matrix variable (list(list(double))).
- `Task/information`: Information items from the optimizer.
  - `int32`: int32 information items (dictionary). A dictionary with entries of the form `name: value`.
  - `int64`: int64 information items (dictionary). A dictionary with entries of the form `name: value`.
  - `double`: double information items (dictionary). A dictionary with entries of the form `name: value`.

16.7.3 A `jtask` example

Listing 16.5: A formatted `jtask` file for a simple portfolio optimization problem.

```
{
   "$schema":"http://mosek.com/json/schema#",
   "Task/name":"Markowitz portfolio with market impact",
   "Task/INFO":{"numvar":7,"numcon":1,"numcone":0,"numbarvar":0,"numanz":6,"numsymmat
↪":0,"numafe":13,"numfnz":12,"numacc":4,"numdjc":0,"numdom":3,"mosekver":[10,0,0,3]},
   "Task/data":{
      "var":{
         "name":["1.0","x[0]","x[1]","x[2]","t[0]","t[1]","t[2]"],
         "bk":["fx","lo","lo","lo","fr","fr","fr"],
         "bl":[1,0.0,0.0,0.0,-1e+30,-1e+30,-1e+30],
         "bu":[1,1e+30,1e+30,1e+30,1e+30,1e+30,1e+30],
         "type":["cont","cont","cont","cont","cont","cont","cont"]
      },
      "con":{
         "name":["budget[]"],
         "bk":["fx"],
         "bl":[1],
```

(continues on next page)

(continued from previous page)


```
         "bu":[1]
      },
      "objective":{
         "sense":"max",
         "name":"obj",
         "c":{
            "subj":[1,2,3],
            "val":[0.1073,0.0737,0.0627]
         },
         "cfix":0.0
      },
      "A":{
         "subi":[0,0,0,0,0,0],
         "subj":[1,2,3,4,5,6],
         "val":[1,1,1,0.01,0.01,0.01]
      },
      "AFE":{
         "numafe":13,
         "F":{
            "subi":[1,1,1,2,2,3,4,6,7,9,10,12],
            "subj":[1,2,3,2,3,3,4,1,5,2,6,3],
            "val":[0.166673333200005,0.0232190712557243,0.0012599496030238,0.
↪102863378954911,-0.00222873156550421,0.0338148677744977,1,1,1,1,1,1]
         },
         "g":{
            "subi":[0,5,8,11],
            "val":[0.035,1,1,1]
         }
      },
      "domains":{
         "type":[["r",0],
                 ["quad",4],
                 ["ppow",3,[0.6666666666666666,0.3333333333333337]]]
      },
      "ACC":{
         "name":["risk[]","tz[0]","tz[1]","tz[2]"],
         "domain":[1,2,2,2],
         "afeidx":[[0,1,2,3],
                   [4,5,6],
                   [7,8,9],
                   [10,11,12]]
      }
   },
   "Task/solutions":{
      "interior":{
         "prosta":"unknown",
         "solsta":"unknown",
         "skx":["fix","supbas","supbas","supbas","supbas","supbas","supbas"],
         "skc":["fix"],
         "xx":[1,0.10331580274282556,0.11673185566457132,0.7724326587076371,0.
↪033208600335718846,0.03988270849469869,0.6788769587942524],
         "xc":[1],
         "slx":[0.0,-5.585840467641202e-10,-8.945844685006369e-10,-7.815248786428623e-
↪11,0.0,0.0,0.0],
         "sux":[0.0,0.0,0.0,0.0,0.0,0.0,0.0],
         "snx":[0.0,0.0,0.0,0.0,0.0,0.0,0.0],
```


(continues on next page)

(continued from previous page)

```
         "slc":[0.0],
         "suc":[-0.046725814048521205],
         "y":[0.046725814048521205],
         "doty":[[-0.6062603164682975,0.3620818321879349,0.17817754087278295,0.
↪4524390346223723],
                 [-4.6725842015519993e-4,-7.708781121860897e-6,2.24800624747081e-4],
                 [-4.6725842015519993e-4,-9.268264309496919e-6,2.390390600079771e-4],
                 [-4.6725842015519993e-4,-1.5854982159992136e-4,6.159249331148646e-4]]
      }
   },
   "Task/parameters":{
      "iparam":{
         "LICENSE_DEBUG":"ON",
         "MIO_SEED":422
      },
      "dparam":{
         "MIO_MAX_TIME":100
      },
      "sparam":{
      }
   }
}
```

16.8 The Solution File Format

MOSEK can output solutions to a text file:

- *basis solution file* (extension `.bas`) if the problem is optimized using the simplex optimizer or basis identification is performed,
- *interior solution file* (extension `.sol`) if a problem is optimized using the interior-point optimizer and no basis identification is required,
- *integer solution file* (extension `.int`) if the problem is solved with the mixed-integer optimizer.

All solution files have the format:

```
NAME                : <problem name>
PROBLEM STATUS      : <status of the problem>
SOLUTION STATUS     : <status of the solution>
OBJECTIVE NAME      : <name of the objective function>
PRIMAL OBJECTIVE    : <primal objective value corresponding to the solution>
DUAL OBJECTIVE      : <dual objective value corresponding to the solution>

CONSTRAINTS
INDEX   NAME     AT ACTIVITY    LOWER LIMIT   UPPER LIMIT   DUAL LOWER   DUAL UPPER
?       <name>   ?? <a value>   <a value>     <a value>     <a value>    <a value>

AFFINE CONIC CONSTRAINTS
INDEX   NAME     I          ACTIVITY    DUAL
?       <name>   <a value>  <a value>   <a value>

VARIABLES
INDEX   NAME     AT ACTIVITY    LOWER LIMIT   UPPER LIMIT   DUAL LOWER   DUAL UPPER   ␣
↪[CONIC DUAL]
?       <name>   ?? <a value>   <a value>     <a value>     <a value>    <a value>    ␣
↪[<a value>]
```

(continues on next page)

(continued from previous page)

```
SYMMETRIC MATRIX VARIABLES
INDEX  NAME     I           J             PRIMAL        DUAL
?      <name>   <a value>   <a value>     <a value>     <a value>
```

The fields `?`, `??` and `<>` will be filled with problem and solution specific information as described below. The solution contains sections corresponding to parts of the input. Empty sections may be omitted and fields in `[]` are optional, depending on what type of problem is solved. The notation below follows the **MOSEK** naming convention for parts of the solution as defined in the problem specifications in Sec. 12.

- `HEADER`
  In this section, first the name of the problem is listed and afterwards the problem and solution status are shown. Next the primal and dual objective values are displayed.
- `CONSTRAINTS`
  - `INDEX`: A sequential index assigned to the constraint by **MOSEK**
  - `NAME`: The name of the constraint assigned by the user or autogenerated.
  - `AT`: The status key `bkc` of the constraint as in Table 16.4.
  - `ACTIVITY`: the activity `xc` of the constraint expression.
  - `LOWER LIMIT`: the lower bound `blc` of the constraint.
  - `UPPER LIMIT`: the upper bound `buc` of the constraint.
  - `DUAL LOWER`: the dual multiplier `slc` corresponding to the lower limit on the constraint.
  - `DUAL UPPER`: the dual multiplier `suc` corresponding to the upper limit on the constraint.
- `AFFINE CONIC CONSTRAINTS`
  - `INDEX`: A sequential index assigned to the affine expressions by **MOSEK**
  - `NAME`: The name of the affine conic constraint assigned by the user or autogenerated.
  - `I`: The sequential index of the affine expression in the affine conic constraint.
  - `ACTIVITY`: the activity of the `I`-th affine expression in the affine conic constraint.
  - `DUAL`: the dual multiplier `doty` for the `I`-th entry in the affine conic constraint.
- `VARIABLES`
  - `INDEX`: A sequential index assigned to the variable by **MOSEK**
  - `NAME`: The name of the variable assigned by the user or autogenerated.
  - `AT`: The status key `bkx` of the variable as in Table 16.4.
  - `ACTIVITY`: the value `xx` of the variable.
  - `LOWER LIMIT`: the lower bound `blx` of the variable.
  - `UPPER LIMIT`: the upper bound `bux` of the variable.
  - `DUAL LOWER`: the dual multiplier `slx` corresponding to the lower limit on the variable.
  - `DUAL UPPER`: the dual multiplier `sux` corresponding to the upper limit on the variable.
  - `CONIC DUAL`: the dual multiplier `skx` corresponding to a conic variable (deprecated).
- `SYMMETRIC MATRIX VARIABLES`
  - `INDEX`: A sequential index assigned to each symmetric matrix entry by **MOSEK**
  - `NAME`: The name of the symmetric matrix variable assigned by the user or autogenerated.
  - `I`: The row index in the symmetric matrix variable.
  - `J`: The column index in the symmetric matrix variable.
  - `PRIMAL`: the value of `barx` for the (`I`, `J`)-th entry in the symmetric matrix variable.

– `DUAL`: the dual multiplier `bars` for the (`I`, `J`)-th entry in the symmetric matrix variable.

Table 16.4: Status keys.

Status key	Interpretation
`UN`	Unknown status
`BS`	Is basic
`SB`	Is superbasic
`LL`	Is at the lower limit (bound)
`UL`	Is at the upper limit (bound)
`EQ`	Lower limit is identical to upper limit
`**`	Is infeasible i.e. the lower limit is greater than the upper limit.

Example.

Below is an example of a solution file.

Listing 16.6: An example of `.sol` file.

```
NAME                 :
PROBLEM STATUS       : PRIMAL_AND_DUAL_FEASIBLE
SOLUTION STATUS      : OPTIMAL
OBJECTIVE NAME       : OBJ
PRIMAL OBJECTIVE     : 0.70571049347734
DUAL OBJECTIVE       : 0.70571048919757

CONSTRAINTS
INDEX      NAME           AT ACTIVITY                LOWER LIMIT        UPPER LIMIT ␣
↪       DUAL LOWER               DUAL UPPER

AFFINE CONIC CONSTRAINTS
INDEX      NAME           I          ACTIVITY                DUAL
0          A1             0          1.0000000009656         0.54475821296644
1          A1             1          0.50000000152223        0.32190455246225
2          A2             0          0.25439922724695        0.4552417870329
3          A2             1          0.17988741850378        -0.32190455246178
4          A2             2          0.17988741850378        -0.32190455246178

VARIABLES
INDEX      NAME           AT ACTIVITY                LOWER LIMIT        UPPER LIMIT ␣
↪       DUAL LOWER               DUAL UPPER
0          X1             SB 0.25439922724695        NONE               NONE        ␣
↪       0                        0
1          X2             SB 0.17988741850378        NONE               NONE        ␣
↪       0                        0
2          X3             SB 0.17988741850378        NONE               NONE        ␣
↪       0                        0

SYMMETRIC MATRIX VARIABLES
INDEX      NAME           I          J          PRIMAL                  DUAL
0          BARX1          0          0          0.21725733689874        1.1333372337141
1          BARX1          1          0          -0.25997257078534       0.
↪67809544651396
2          BARX1          2          0          0.21725733648507        -0.
↪3219045527104
3          BARX1          1          1          0.31108610088839        1.1333372332693
4          BARX1          2          1          -0.25997257078534       0.
```

(continues on next page)

(continued from previous page)

```
↪67809544651435
5           BARX1           2           2           0.21725733689874          1.1333372337145
6           BARX2           0           0           4.8362272828127e-10       0.
↪54475821339698
7           BARX2           1           0           0                         0
8           BARX2           1           1           4.8362272828127e-10       0.
↪54475821339698
```

Chapter 17

List of examples

List of examples shipped in the distribution of Optimizer API for Python:

Table 17.1: List of distributed examples

File	Description
`acc1.py`	A simple problem with one affine conic constraint (ACC)
`acc2.py`	A simple problem with two affine conic constraints (ACC)
`blas_lapack.py`	Demonstrates the **MOSEK** interface to BLAS/LAPACK linear algebra routines
`callback.py`	An example of data/progress callback
`ceo1.py`	A simple conic exponential problem
`concurrent1.py`	Implementation of a concurrent optimizer for linear and mixed-integer problems
`cqo1.py`	A simple conic quadratic problem
`djc1.py`	A simple problem with disjunctive constraints (DJC)
`feasrepairex1.py`	A simple example of how to repair an infeasible problem
`gp1.py`	A simple geometric program (GP) in conic form
`helloworld.py`	A Hello World example
`lo1.py`	A simple linear problem
`lo2.py`	A simple linear problem
`logistic.py`	Implements logistic regression and simple log-sum-exp (CEO)
`mico1.py`	A simple mixed-integer conic problem
`milo1.py`	A simple mixed-integer linear problem
`mioinitsol.py`	A simple mixed-integer linear problem with an initial guess
`opt_server_async.py`	Uses **MOSEK** OptServer to solve an optimization problem asynchronously
`opt_server_sync.py`	Uses **MOSEK** OptServer to solve an optimization problem synchronously
`parallel.py`	Demonstrates parallel optimization using a batch method in MOSEK
`parameters.py`	Shows how to set optimizer parameters and read information items
`pinfeas.py`	Shows how to obtain and analyze a primal infeasibility certificate
`portfolio_1_basi.py`	Portfolio optimization - basic Markowitz model
`portfolio_2_fron.py`	Portfolio optimization - efficient frontier
`portfolio_3_impa.py`	Portfolio optimization - market impact costs
`portfolio_4_tran.py`	Portfolio optimization - transaction costs
`portfolio_5_card.py`	Portfolio optimization - cardinality constraints
`portfolio_6_fact.py`	Portfolio optimization - factor model
`pow1.py`	A simple power cone problem

continues on next page

Table 17.1 – continued from previous page

File	Description
qcqo1.py	A simple quadratically constrained quadratic problem
qo1.py	A simple quadratic problem
reoptimization.py	Demonstrate how to modify and re-optimize a linear problem
response.py	Demonstrates proper response handling
sdo1.py	A simple semidefinite problem with one matrix variable and a quadratic cone
sdo2.py	A simple semidefinite problem with two matrix variables
sdo_lmi.py	A simple semidefinite problem with an LMI using the SVEC domain.
sensitivity.py	Sensitivity analysis performed on a small linear problem
simple.py	A simple I/O example: read problem from a file, solve and write solutions
solutionquality.py	Demonstrates how to examine the quality of a solution
solvebasis.py	Demonstrates solving a linear system with the basis matrix
solvelinear.py	Demonstrates solving a general linear system
sparsecholesky.py	Shows how to find a Cholesky factorization of a sparse matrix

Additional examples can be found on the **MOSEK** website and in other **MOSEK** publications.

Chapter 18

Interface changes

The section shows interface-specific changes to the **MOSEK** Optimizer API for Python in version 11.2 compared to version 10. See the release notes for general changes and new features of the **MOSEK** Optimization Suite.

18.1 Important changes compared to version 10

- **Parameters.** Users who set parameters to tune the performance and numerical properties of the solver (termination criteria, tolerances, solving primal or dual, presolve etc.) are recommended to reevaluate such tuning. It may be that other, or default, parameter settings will be more beneficial in the current version. The hints in Sec. 8 may be useful for some cases.
- **Data types.** The `mosek` module will issue a warning if data is passed as a numpy array of type `int64` where `int32` is expected, or the other way around. In this case the module needs to make an internal copy before calling the solver and this extra step could be eliminated by the user passing data of correct type from the start.
- Renamed `getnumanz64` to `getnumanz` and `setdefaults` to `resetparameters`.

18.2 Changes compared to version 10

18.2.1 Functions compared to version 10

Added

- *Task.appenddualpowerconedomainseq*
- *Task.appendprimalpowerconedomainseq*
- *Task.getdualproblem*
- *Task.getlintparam*
- *Task.putlintparam*
- *Task.resetdouparam*
- *Task.resetintparam*
- *Task.resetparameters*
- *Task.resetstrparam*

Removed

- Task.setdefaults

18.2.2 Parameters compared to version 10

Added

- *dparam.folding_tol_eq*
- *dparam.mio_clique_table_size_factor*
- *dparam.sim_precision_scaling_extended*
- *dparam.sim_precision_scaling_normal*
- *iparam.folding_use*
- *iparam.getdual_convert_lmis*
- *iparam.heartbeat_sim_freq_ticks*
- *iparam.log_sim_freq_giga_ticks*
- *iparam.mio_conflict_analysis_level*
- *iparam.mio_crossover_max_nodes*
- *iparam.mio_independent_block_level*
- *iparam.mio_opt_face_max_nodes*
- *iparam.mio_rens_max_nodes*
- *iparam.ptf_write_single_psd_terms*
- *iparam.read_async*
- *iparam.sim_precision*
- *iparam.sim_precision_boost*
- *iparam.write_async*

Removed

- dparam.check_convexity_rel_tol
- dparam.presolve_tol_aij
- iparam.infeas_prefer_primal
- iparam.intpnt_max_num_refinement_steps
- iparam.intpnt_purify
- iparam.log_response
- iparam.log_sim_minor
- iparam.mio_root_repeat_presolve_level
- iparam.presolve_level
- iparam.sensitivity_optimizer
- iparam.sim_stability_priority

- iparam.sol_filter_keep_ranged
- iparam.solution_callback
- iparam.write_data_param
- iparam.write_generic_names_io
- iparam.write_task_inc_sol
- iparam.write_xml_mode
- sparam.write_lp_gen_var_name

18.2.3 Constants compared to version 10

Added

- *callbackcode.begin_folding*
- *callbackcode.begin_folding_bi*
- *callbackcode.begin_folding_bi_dual*
- *callbackcode.begin_folding_bi_initialize*
- *callbackcode.begin_folding_bi_optimizer*
- *callbackcode.begin_folding_bi_primal*
- *callbackcode.begin_initialize_bi*
- *callbackcode.begin_optimize_bi*
- *callbackcode.decomp_mio*
- *callbackcode.end_folding*
- *callbackcode.end_folding_bi*
- *callbackcode.end_folding_bi_dual*
- *callbackcode.end_folding_bi_initialize*
- *callbackcode.end_folding_bi_optimizer*
- *callbackcode.end_folding_bi_primal*
- *callbackcode.end_initialize_bi*
- *callbackcode.end_optimize_bi*
- *callbackcode.folding_bi_dual*
- *callbackcode.folding_bi_optimizer*
- *callbackcode.folding_bi_primal*
- *callbackcode.heartbeat*
- *callbackcode.optimize_bi*
- *callbackcode.qo_reformulate*
- *dinfitem.folding_bi_optimize_time*

- *dinfitem.folding_bi_unfold_dual_time*
- *dinfitem.folding_bi_unfold_initialize_time*
- *dinfitem.folding_bi_unfold_primal_time*
- *dinfitem.folding_bi_unfold_time*
- *dinfitem.folding_factor*
- *dinfitem.folding_time*
- *iinfitem.folding_applied*
- *iinfitem.mio_final_numbin*
- *iinfitem.mio_final_numbinconevar*
- *iinfitem.mio_final_numcon*
- *iinfitem.mio_final_numcone*
- *iinfitem.mio_final_numconevar*
- *iinfitem.mio_final_numcont*
- *iinfitem.mio_final_numcontconevar*
- *iinfitem.mio_final_numdexpcones*
- *iinfitem.mio_final_numdjc*
- *iinfitem.mio_final_numdpowcones*
- *iinfitem.mio_final_numint*
- *iinfitem.mio_final_numintconevar*
- *iinfitem.mio_final_numpexpcones*
- *iinfitem.mio_final_numppowcones*
- *iinfitem.mio_final_numqcones*
- *iinfitem.mio_final_numrqcones*
- *iinfitem.mio_final_numvar*
- *iinfitem.mio_num_blocks_solved_in_bb*
- *iinfitem.mio_num_blocks_solved_in_presolve*
- *liinfitem.bi_clean_iter*
- *liinfitem.folding_bi_dual_iter*
- *liinfitem.folding_bi_optimizer_iter*
- *liinfitem.folding_bi_primal_iter*
- *liinfitem.mio_final_anz*
- *optimizertype.new_dual_simplex*
- *optimizertype.new_primal_simplex*

Removed

- constant.callbackcode.begin_simplex_bi
- constant.callbackcode.im_bi
- constant.callbackcode.im_conic
- constant.callbackcode.im_dual_bi
- constant.callbackcode.im_intpnt
- constant.callbackcode.im_presolve
- constant.callbackcode.im_primal_bi
- constant.callbackcode.im_qo_reformulate
- constant.callbackcode.im_simplex_bi
- constant.dinfitem.bi_clean_dual_time
- constant.dinfitem.bi_clean_primal_time
- constant.liinfitem.bi_clean_dual_deg_iter
- constant.liinfitem.bi_clean_dual_iter
- constant.liinfitem.bi_clean_primal_deg_iter
- constant.liinfitem.bi_clean_primal_iter

18.2.4 Response Codes compared to version 10

Added

- *rescode.err_getdual_not_available*
- *rescode.err_read_async*
- *rescode.err_read_premature_eof*
- *rescode.err_read_write*
- *rescode.err_server_hard_timeout*
- *rescode.err_task_premature_eof*
- *rescode.err_write_async*
- *rescode.err_write_lp_duplicate_con_names*
- *rescode.err_write_lp_duplicate_var_names*
- *rescode.err_write_lp_invalid_con_names*
- *rescode.err_write_lp_invalid_var_names*
- *rescode.trm_server_max_memory*
- *rescode.trm_server_max_time*
- *rescode.wrn_getdual_ignores_integrality*
- *rescode.wrn_presolve_primal_perturbations*
- *rescode.wrn_ptf_unknown_section*

Removed

- rescode.err_invalid_ampl_stub
- rescode.err_size_license_numcores
- rescode.err_xml_invalid_problem_type
- rescode.wrn_presolve_primal_pertubations
- rescode.wrn_write_lp_duplicate_con_names
- rescode.wrn_write_lp_duplicate_var_names
- rescode.wrn_write_lp_invalid_con_names
- rescode.wrn_write_lp_invalid_var_names

Bibliography

[AA95] E. D. Andersen and K. D. Andersen. Presolving in linear programming. *Math. Programming*, 71(2):221–245, 1995.

[AGMeszarosX96] E. D. Andersen, J. Gondzio, Cs. Mészáros, and X. Xu. Implementation of interior point methods for large scale linear programming. In T. Terlaky, editor, *Interior-point methods of mathematical programming*, pages 189–252. Kluwer Academic Publishers, 1996.

[ART03] E. D. Andersen, C. Roos, and T. Terlaky. On implementing a primal-dual interior-point method for conic quadratic optimization. *Math. Programming*, February 2003.

[AY96] E. D. Andersen and Y. Ye. Combining interior-point and pivoting algorithms. *Management Sci.*, 42(12):1719–1731, December 1996.

[And09] Erling D. Andersen. The homogeneous and self-dual model and algorithm for linear optimization. Technical Report TR-1-2009, MOSEK ApS, 2009. URL: http://docs.mosek.com/whitepapers/homolo.pdf.

[And13] Erling D. Andersen. On formulating quadratic functions in optimization models. Technical Report TR-1-2013, MOSEK ApS, 2013. Last revised 23-feb-2016. URL: http://docs.mosek.com/whitepapers/qmodel.pdf.

[BKVH07] S. Boyd, S.J. Kim, L. Vandenberghe, and A. Hassibi. A Tutorial on Geometric Programming. *Optimization and Engineering*, 8(1):67–127, 2007. Available at http://www.stanford.edu/\protect\unhbox\voidb@x\penalty\@M\boyd/gp_tutorial.html.

[Chvatal83] V. Chvátal. *Linear programming*. W.H. Freeman and Company, 1983.

[CCornuejolsZ14] M. Conforti, G. Cornu/'ejols, and G. Zambelli. *Integer programming*. Springer, 2014.

[GK00] Richard C. Grinold and Ronald N. Kahn. *Active portfolio management*. McGraw-Hill, New York, 2 edition, 2000.

[Naz87] J. L. Nazareth. *Computer Solution of Linear Programs*. Oxford University Press, New York, 1987.

[RTV97] C. Roos, T. Terlaky, and J. -Ph. Vial. *Theory and algorithms for linear optimization: an interior point approach*. John Wiley and Sons, New York, 1997.

[Ste98] G. W. Stewart. *Matrix Algorithms. Volume 1: Basic decompositions*. SIAM, 1998.

[Wal00] S. W. Wallace. Decision making under uncertainty: is sensitivity of any use. *Oper. Res.*, 48(1):20–25, January 2000.

[Wol98] L. A. Wolsey. *Integer programming*. John Wiley and Sons, 1998.

Symbol Index

Classes

Env, 221
Env.Task, 221
Env.syrk, 230
Env.syevd, 229
Env.syeig, 229
Env.sparsetriangularsolvedense, 229
Env.set_Stream, 228
Env.resetexpirylicenses, 228
Env.putlicensewait, 228
Env.putlicensepath, 228
Env.putlicensedebug, 228
Env.putlicensecode, 227
Env.potrf, 227
Env.optimizebatch, 226
Env.linkfiletostream, 226
Env.licensecleanup, 226
Env.getversion, 226
Env.getcodedesc, 226
Env.gemv, 225
Env.gemm, 224
Env.expirylicenses, 224
Env.Env, 221
Env.echointro, 224
Env.dot, 224
Env.computesparsecholesky, 222
Env.checkoutlicense, 222
Env.checkinlicense, 222
Env.checkinall, 222
Env.axpy, 221
Env.__del__, 221
Task, 231
Task.writetask, 356
Task.writesolutionfile, 356
Task.writesolution, 355
Task.writeparamfile, 355
Task.writejsonsol, 355
Task.writedatastream, 355
Task.writedata, 354
Task.writebsolution, 354
Task.updatesolutioninfo, 354
Task.toconic, 354
Task.Task, 231
Task.strtosk, 353
Task.strtoconetype, 353
Task.solvewithbasis, 352
Task.solutionsummary, 352
Task.solutiondef, 352
Task.set_Stream, 352
Task.set_Progress, 352
Task.set_InfoCallback, 351
Task.sensitivityreport, 351
Task.resizetask, 351
Task.resetstrparam, 350
Task.resetparameters, 350
Task.resetintparam, 350
Task.resetdouparam, 350
Task.removevars, 350
Task.removecons, 350
Task.removecones, 349
Task.removebarvars, 349
Task.readtask, 349
Task.readsummary, 349
Task.readsolutionfile, 348
Task.readsolution, 348
Task.readptfstring, 348
Task.readparamfile, 348
Task.readopfstring, 347
Task.readlpstring, 347
Task.readjsonstring, 347
Task.readjsonsol, 347
Task.readdataformat, 347
Task.readdata, 346
Task.readbsolution, 346
Task.putyslice, 346
Task.puty, 346
Task.putxxslice, 346
Task.putxx, 345
Task.putxcslice, 345
Task.putxc, 345
Task.putvartypelist, 345
Task.putvartype, 344
Task.putvarsolutionj, 344
Task.putvarname, 344
Task.putvarboundsliceconst, 344
Task.putvarboundslice, 343
Task.putvarboundlistconst, 343
Task.putvarboundlist, 343
Task.putvarbound, 342
Task.puttaskname, 342
Task.putsuxslice, 342
Task.putsux, 342
Task.putsucslice, 342
Task.putsuc, 341
Task.putstrparam, 341
Task.putsolutionyi, 341
Task.putsolutionnew, 340

Task.putsolution, 340
Task.putsnxslice, 339
Task.putsnx, 339
Task.putslxslice, 339
Task.putslx, 339
Task.putslcslice, 338
Task.putslc, 338
Task.putskxslice, 338
Task.putskx, 338
Task.putskcslice, 337
Task.putskc, 337
Task.putqobjij, 337
Task.putqobj, 336
Task.putqconk, 336
Task.putqcon, 336
Task.putparam, 335
Task.putoptserverhost, 335
Task.putobjsense, 335
Task.putobjname, 335
Task.putnastrparam, 334
Task.putnaintparam, 334
Task.putnadouparam, 334
Task.putmaxnumvar, 334
Task.putmaxnumqnz, 333
Task.putmaxnumdomain, 333
Task.putmaxnumdjc, 333
Task.putmaxnumcone, 333
Task.putmaxnumcon, 332
Task.putmaxnumbarvar, 332
Task.putmaxnumanz, 332
Task.putmaxnumafe, 331
Task.putmaxnumacc, 331
Task.putlintparam, 331
Task.putintparam, 331
Task.putdouparam, 331
Task.putdomainname, 330
Task.putdjcslice, 330
Task.putdjcname, 330
Task.putdjc, 329
Task.putcslice, 329
Task.putconsolutioni, 328
Task.putconname, 328
Task.putconename, 328
Task.putcone, 328
Task.putconboundsliceconst, 327
Task.putconboundslice, 327
Task.putconboundlistconst, 327
Task.putconboundlist, 327
Task.putconbound, 326
Task.putclist, 326
Task.putcj, 326
Task.putcfix, 325
Task.putbarxj, 325
Task.putbarvarname, 325
Task.putbarsj, 325
Task.putbarcj, 325
Task.putbarcblocktriplet, 324
Task.putbararowlist, 324
Task.putbaraijlist, 323
Task.putbaraij, 323
Task.putbarablocktriplet, 323
Task.putatruncatetol, 322
Task.putarowslice, 322
Task.putarowlist, 322
Task.putarow, 321
Task.putaijlist, 321
Task.putaij, 321
Task.putafegslice, 321
Task.putafeglist, 320
Task.putafeg, 320
Task.putafefrowlist, 320
Task.putafefrow, 319
Task.putafefentrylist, 319
Task.putafefentry, 319
Task.putafefcol, 319
Task.putafebarfrow, 318
Task.putafebarfentrylist, 317
Task.putafebarfentry, 317
Task.putafebarfblocktriplet, 317
Task.putacolslice, 316
Task.putacollist, 316
Task.putacol, 316
Task.putaccname, 315
Task.putacclist, 315
Task.putaccdoty, 315
Task.putaccbj, 315
Task.putaccb, 314
Task.putacc, 314
Task.primalsensitivity, 313
Task.primalrepair, 312
Task.optimizersummary, 312
Task.optimizermt, 311
Task.optimize, 311
Task.onesolutionsummary, 311
Task.linkfiletostream, 311
Task.isstrparname, 311
Task.isintparname, 310
Task.isdouparname, 310
Task.inputdata, 309
Task.initbasissolve, 309
Task.infeasibilityreport, 309
Task.getyslice, 308
Task.gety, 308
Task.getxxslice, 308
Task.getxx, 307
Task.getxcslice, 307
Task.getxc, 307
Task.getvartypelist, 307
Task.getvartype, 306
Task.getvarnamelen, 306
Task.getvarnameindex, 306
Task.getvarname, 306
Task.getvarboundslice, 305
Task.getvarbound, 305
Task.gettasknamelen, 305
Task.gettaskname, 305

Task.getsymmatinfo, 304
Task.getsuxslice, 304
Task.getsux, 304
Task.getsucslice, 303
Task.getsuc, 303
Task.getstrparamlen, 303
Task.getstrparam, 303
Task.getsparsesymmat, 302
Task.getsolutionslice, 302
Task.getsolutionnew, 300
Task.getsolutioninfonew, 299
Task.getsolutioninfo, 299
Task.getsolution, 296
Task.getsolsta, 296
Task.getsnxslice, 296
Task.getsnx, 296
Task.getslxslice, 295
Task.getslx, 295
Task.getslcslice, 295
Task.getslc, 294
Task.getskxslice, 294
Task.getskx, 294
Task.getskn, 293
Task.getskcslice, 293
Task.getskc, 293
Task.getreducedcosts, 292
Task.getqobjij, 292
Task.getqobj, 292
Task.getqconk, 291
Task.getpviolvar, 291
Task.getpvioldjc, 291
Task.getpviolcones, 290
Task.getpviolcon, 290
Task.getpviolbarvar, 289
Task.getpviolacc, 289
Task.getprosta, 288
Task.getprobtype, 288
Task.getprimalsolutionnorms, 288
Task.getprimalobj, 288
Task.getpowerdomaininfo, 288
Task.getpowerdomainalpha, 287
Task.getobjsense, 287
Task.getobjnamelen, 287
Task.getobjname, 287
Task.getnumvar, 287
Task.getnumsymmat, 286
Task.getnumqobjnz, 286
Task.getnumqconknz, 286
Task.getnumparam, 286
Task.getnumintvar, 286
Task.getnumdomain, 285
Task.getnumdjc, 285
Task.getnumconemem, 285
Task.getnumcone, 285
Task.getnumcon, 285
Task.getnumbarvar, 284
Task.getnumbarcnz, 284
Task.getnumbarcblocktriplets, 284
Task.getnumbaranz, 284
Task.getnumbarablocktriplets, 284
Task.getnumanz, 283
Task.getnumafe, 283
Task.getnumacc, 283
Task.getmemusage, 283
Task.getmaxnumvar, 283
Task.getmaxnumqnz, 282
Task.getmaxnumcone, 282
Task.getmaxnumcon, 282
Task.getmaxnumbarvar, 282
Task.getmaxnumanz, 282
Task.getlintparam, 281
Task.getlintinf, 281
Task.getlenbarvarj, 281
Task.getintparam, 281
Task.getintinf, 280
Task.getinfeasiblesubproblem, 280
Task.getdviolvar, 280
Task.getdviolcones, 279
Task.getdviolcon, 279
Task.getdviolbarvar, 278
Task.getdviolacc, 278
Task.getdualsolutionnorms, 277
Task.getdualproblem, 277
Task.getdualobj, 277
Task.getdouparam, 276
Task.getdouinf, 276
Task.getdomaintype, 276
Task.getdomainnamelen, 276
Task.getdomainname, 276
Task.getdomainn, 275
Task.getdjctermsizelist, 275
Task.getdjcs, 274
Task.getdjcnumtermtot, 274
Task.getdjcnumterm, 274
Task.getdjcnumdomaintot, 274
Task.getdjcnumdomain, 274
Task.getdjcnumafetot, 273
Task.getdjcnumafe, 273
Task.getdjcnamelen, 273
Task.getdjcname, 273
Task.getdjcdomainidxlist, 272
Task.getdjcb, 272
Task.getdjcafeidxlist, 272
Task.getdimbarvarj, 272
Task.getcslice, 271
Task.getconnamelen, 271
Task.getconnameindex, 271
Task.getconname, 271
Task.getconenamelen, 270
Task.getconenameindex, 270
Task.getconename, 270
Task.getconeinfo, 270
Task.getcone, 269
Task.getconboundslice, 269
Task.getconbound, 268
Task.getclist, 268

Task.getcj, 268
Task.getcfix, 268
Task.getc, 267
Task.getbarxslice, 267
Task.getbarxj, 267
Task.getbarvarnamelen, 267
Task.getbarvarnameindex, 266
Task.getbarvarname, 266
Task.getbarsslice, 266
Task.getbarsj, 265
Task.getbarcsparsity, 265
Task.getbarcidxj, 265
Task.getbarcidxinfo, 265
Task.getbarcidx, 264
Task.getbarcblocktriplet, 264
Task.getbarasparsity, 263
Task.getbaraidxinfo, 263
Task.getbaraidxij, 263
Task.getbaraidx, 262
Task.getbarablocktriplet, 262
Task.getatruncatetol, 261
Task.getatrip, 261
Task.getarowslicetrip, 261
Task.getarowslicenumnz, 260
Task.getarowslice, 260
Task.getarownumnz, 260
Task.getarow, 259
Task.getapiecenumnz, 259
Task.getaij, 259
Task.getafegslice, 258
Task.getafeg, 258
Task.getafeftrip, 258
Task.getafefrownumnz, 258
Task.getafefrow, 257
Task.getafefnumnz, 257
Task.getafebarfrowinfo, 257
Task.getafebarfrow, 256
Task.getafebarfnumrowentries, 256
Task.getafebarfnumblocktriplets, 256
Task.getafebarfblocktriplet, 255
Task.getacolslicetrip, 255
Task.getacolslicenumnz, 254
Task.getacolslice, 254
Task.getacolnumnz, 254
Task.getacol, 253
Task.getaccs, 253
Task.getaccntot, 253
Task.getaccnamelen, 252
Task.getaccname, 252
Task.getaccn, 252
Task.getaccgvector, 252
Task.getaccftrip, 251
Task.getaccfnumnz, 251
Task.getaccdotys, 251
Task.getaccdoty, 250
Task.getaccdomain, 250
Task.getaccbarfnumblocktriplets, 250
Task.getaccbarfblocktriplet, 249
Task.getaccb, 249
Task.getaccafeidxlist, 249
Task.generatevarnames, 249
Task.generatedjcnames, 248
Task.generateconnames, 248
Task.generateconenames, 248
Task.generatebarvarnames, 247
Task.generateaccnames, 247
Task.evaluateaccs, 247
Task.evaluateacc, 246
Task.emptyafefrowlist, 246
Task.emptyafefrow, 246
Task.emptyafefcollist, 246
Task.emptyafefcol, 246
Task.emptyafebarfrowlist, 246
Task.emptyafebarfrow, 245
Task.dualsensitivity, 245
Task.deletesolution, 244
Task.commitchanges, 244
Task.chgvarbound, 244
Task.chgconbound, 243
Task.checkmem, 243
Task.basiscond, 243
Task.asyncstop, 243
Task.asyncpoll, 242
Task.asyncoptimize, 242
Task.asyncgetresult, 242
Task.appendvars, 241
Task.appendsvecpsdconedomain, 241
Task.appendsparsesymmatlist, 240
Task.appendsparsesymmat, 240
Task.appendrzerodomain, 240
Task.appendrquadraticconedomain, 239
Task.appendrplusdomain, 239
Task.appendrminusdomain, 239
Task.appendrdomain, 239
Task.appendquadraticconedomain, 239
Task.appendprimalpowerconedomainseq, 238
Task.appendprimalpowerconedomain, 238
Task.appendprimalgeomeanconedomain, 238
Task.appendprimalexpconedomain, 237
Task.appenddualpowerconedomainseq, 237
Task.appenddualpowerconedomain, 237
Task.appenddualgeomeanconedomain, 236
Task.appenddualexpconedomain, 236
Task.appenddjcs, 236
Task.appendcons, 236
Task.appendconesseq, 235
Task.appendconeseq, 235
Task.appendcone, 234
Task.appendbarvars, 234
Task.appendafes, 234
Task.appendaccsseq, 233
Task.appendaccseq, 233
Task.appendaccs, 232
Task.appendacc, 232
Task.analyzesolution, 232
Task.analyzeproblem, 231

Task.analyzenames, 231
Task.__del__, 231

Enumerations

basindtype, 464
basindtype.reserved, 464
basindtype.no_error, 464
basindtype.never, 464
basindtype.if_feasible, 464
basindtype.always, 464
boundkey, 465
boundkey.up, 465
boundkey.ra, 465
boundkey.lo, 465
boundkey.fx, 465
boundkey.fr, 465
branchdir, 488
branchdir.up, 488
branchdir.root_lp, 488
branchdir.pseudocost, 488
branchdir.near, 488
branchdir.guided, 488
branchdir.free, 488
branchdir.far, 488
branchdir.down, 488
callbackcode, 466
callbackcode.write_opf, 472
callbackcode.update_simplex, 472
callbackcode.update_primal_simplex_bi, 471
callbackcode.update_primal_simplex, 471
callbackcode.update_primal_bi, 471
callbackcode.update_presolve, 471
callbackcode.update_dual_simplex_bi, 471
callbackcode.update_dual_simplex, 471
callbackcode.update_dual_bi, 471
callbackcode.solving_remote, 471
callbackcode.restart_mio, 471
callbackcode.read_opf_section, 471
callbackcode.read_opf, 471
callbackcode.qo_reformulate, 471
callbackcode.primal_simplex, 471
callbackcode.optimize_bi, 471
callbackcode.new_int_mio, 471
callbackcode.intpnt, 471
callbackcode.im_simplex, 471
callbackcode.im_root_cutgen, 471
callbackcode.im_read, 471
callbackcode.im_primal_simplex, 471
callbackcode.im_primal_sensivity, 470
callbackcode.im_order, 470
callbackcode.im_mio_primal_simplex, 470
callbackcode.im_mio_intpnt, 470
callbackcode.im_mio_dual_simplex, 470
callbackcode.im_mio, 470
callbackcode.im_lu, 470
callbackcode.im_license_wait, 470
callbackcode.im_dual_simplex, 470
callbackcode.im_dual_sensivity, 470
callbackcode.heartbeat, 470
callbackcode.folding_bi_primal, 470
callbackcode.folding_bi_optimizer, 470
callbackcode.folding_bi_dual, 470
callbackcode.end_write, 470
callbackcode.end_to_conic, 470
callbackcode.end_solve_root_relax, 470
callbackcode.end_simplex_bi, 470
callbackcode.end_simplex, 470
callbackcode.end_root_cutgen, 470
callbackcode.end_read, 469
callbackcode.end_qcqo_reformulate, 469
callbackcode.end_primal_simplex_bi, 469
callbackcode.end_primal_simplex, 469
callbackcode.end_primal_setup_bi, 469
callbackcode.end_primal_sensitivity, 469
callbackcode.end_primal_repair, 469
callbackcode.end_primal_bi, 469
callbackcode.end_presolve, 469
callbackcode.end_optimizer, 469
callbackcode.end_optimize_bi, 469
callbackcode.end_mio, 469
callbackcode.end_license_wait, 469
callbackcode.end_intpnt, 469
callbackcode.end_initialize_bi, 469
callbackcode.end_infeas_ana, 469
callbackcode.end_folding_bi_primal, 469
callbackcode.end_folding_bi_optimizer, 469
callbackcode.end_folding_bi_initialize, 469
callbackcode.end_folding_bi_dual, 469
callbackcode.end_folding_bi, 469
callbackcode.end_folding, 468
callbackcode.end_dual_simplex_bi, 468
callbackcode.end_dual_simplex, 468
callbackcode.end_dual_setup_bi, 468
callbackcode.end_dual_sensitivity, 468
callbackcode.end_dual_bi, 468
callbackcode.end_conic, 468
callbackcode.end_bi, 468
callbackcode.dual_simplex, 468
callbackcode.decomp_mio, 468
callbackcode.conic, 468
callbackcode.begin_write, 468
callbackcode.begin_to_conic, 468
callbackcode.begin_solve_root_relax, 468
callbackcode.begin_simplex, 468
callbackcode.begin_root_cutgen, 468
callbackcode.begin_read, 468
callbackcode.begin_qcqo_reformulate, 468
callbackcode.begin_primal_simplex_bi, 468
callbackcode.begin_primal_simplex, 468
callbackcode.begin_primal_setup_bi, 468
callbackcode.begin_primal_sensitivity, 467
callbackcode.begin_primal_repair, 467
callbackcode.begin_primal_bi, 467
callbackcode.begin_presolve, 467
callbackcode.begin_optimizer, 467
callbackcode.begin_optimize_bi, 467

callbackcode.begin_mio, 467
callbackcode.begin_license_wait, 467
callbackcode.begin_intpnt, 467
callbackcode.begin_initialize_bi, 467
callbackcode.begin_infeas_ana, 467
callbackcode.begin_folding_bi_primal, 467
callbackcode.begin_folding_bi_optimizer, 467
callbackcode.begin_folding_bi_initialize, 467
callbackcode.begin_folding_bi_dual, 467
callbackcode.begin_folding_bi, 467
callbackcode.begin_folding, 467
callbackcode.begin_dual_simplex_bi, 467
callbackcode.begin_dual_simplex, 467
callbackcode.begin_dual_setup_bi, 467
callbackcode.begin_dual_sensitivity, 467
callbackcode.begin_dual_bi, 466
callbackcode.begin_conic, 466
callbackcode.begin_bi, 466
compresstype, 472
compresstype.zstd, 472
compresstype.none, 472
compresstype.gzip, 472
compresstype.free, 472
conetype, 472
conetype.zero, 472
conetype.rquad, 472
conetype.quad, 472
conetype.ppow, 472
conetype.pexp, 472
conetype.dpow, 472
conetype.dexp, 472
dataformat, 473
dataformat.task, 473
dataformat.ptf, 473
dataformat.op, 473
dataformat.mps, 473
dataformat.lp, 473
dataformat.json_task, 473
dataformat.free_mps, 473
dataformat.extension, 473
dataformat.cb, 473
dinfitem, 474
dinfitem.write_data_time, 480
dinfitem.to_conic_time, 480
dinfitem.sol_itr_pviolvar, 480
dinfitem.sol_itr_pviolcones, 480
dinfitem.sol_itr_pviolcon, 480
dinfitem.sol_itr_pviolbarvar, 480
dinfitem.sol_itr_pviolacc, 479
dinfitem.sol_itr_primal_obj, 479
dinfitem.sol_itr_nrm_y, 479
dinfitem.sol_itr_nrm_xx, 479
dinfitem.sol_itr_nrm_xc, 479
dinfitem.sol_itr_nrm_sux, 479
dinfitem.sol_itr_nrm_suc, 479
dinfitem.sol_itr_nrm_snx, 479
dinfitem.sol_itr_nrm_slx, 479
dinfitem.sol_itr_nrm_slc, 479
dinfitem.sol_itr_nrm_barx, 479
dinfitem.sol_itr_nrm_bars, 479
dinfitem.sol_itr_dviolvar, 479
dinfitem.sol_itr_dviolcones, 479
dinfitem.sol_itr_dviolcon, 479
dinfitem.sol_itr_dviolbarvar, 479
dinfitem.sol_itr_dviolacc, 479
dinfitem.sol_itr_dual_obj, 479
dinfitem.sol_itg_pviolvar, 479
dinfitem.sol_itg_pviolitg, 478
dinfitem.sol_itg_pvioldjc, 478
dinfitem.sol_itg_pviolcones, 478
dinfitem.sol_itg_pviolcon, 478
dinfitem.sol_itg_pviolbarvar, 478
dinfitem.sol_itg_pviolacc, 478
dinfitem.sol_itg_primal_obj, 478
dinfitem.sol_itg_nrm_xx, 478
dinfitem.sol_itg_nrm_xc, 478
dinfitem.sol_itg_nrm_barx, 478
dinfitem.sol_bas_pviolvar, 478
dinfitem.sol_bas_pviolcon, 478
dinfitem.sol_bas_primal_obj, 478
dinfitem.sol_bas_nrm_y, 478
dinfitem.sol_bas_nrm_xx, 478
dinfitem.sol_bas_nrm_xc, 478
dinfitem.sol_bas_nrm_sux, 478
dinfitem.sol_bas_nrm_suc, 478
dinfitem.sol_bas_nrm_slx, 478
dinfitem.sol_bas_nrm_slc, 477
dinfitem.sol_bas_nrm_barx, 477
dinfitem.sol_bas_dviolvar, 477
dinfitem.sol_bas_dviolcon, 477
dinfitem.sol_bas_dual_obj, 477
dinfitem.sim_time, 477
dinfitem.sim_primal_time, 477
dinfitem.sim_obj, 477
dinfitem.sim_feas, 477
dinfitem.sim_dual_time, 477
dinfitem.remote_time, 477
dinfitem.read_data_time, 477
dinfitem.qcqo_reformulate_worst_cholesky_diag_scaling, 477
dinfitem.qcqo_reformulate_worst_cholesky_column_scaling, 477
dinfitem.qcqo_reformulate_time, 477
dinfitem.qcqo_reformulate_max_perturbation, 477
dinfitem.primal_repair_penalty_obj, 477
dinfitem.presolve_total_primal_perturbation, 477
dinfitem.presolve_time, 477
dinfitem.presolve_lindep_time, 477
dinfitem.presolve_eli_time, 477
dinfitem.optimizer_time, 476
dinfitem.optimizer_ticks, 476
dinfitem.mio_user_obj_cut, 476

dinfitem.mio_time, 476
dinfitem.mio_symmetry_factor, 476
dinfitem.mio_symmetry_detection_time, 476
dinfitem.mio_root_time, 476
dinfitem.mio_root_presolve_time, 476
dinfitem.mio_root_optimizer_time, 476
dinfitem.mio_root_cut_separation_time, 476
dinfitem.mio_root_cut_selection_time, 476
dinfitem.mio_probing_time, 476
dinfitem.mio_obj_rel_gap, 476
dinfitem.mio_obj_int, 476
dinfitem.mio_obj_bound, 476
dinfitem.mio_obj_abs_gap, 476
dinfitem.mio_lipro_separation_time, 475
dinfitem.mio_lipro_selection_time, 475
dinfitem.mio_knapsack_cover_separation_time, 475
dinfitem.mio_knapsack_cover_selection_time, 475
dinfitem.mio_initial_feasible_solution_obj, 475
dinfitem.mio_implied_bound_separation_time, 475
dinfitem.mio_implied_bound_selection_time, 475
dinfitem.mio_gmi_separation_time, 475
dinfitem.mio_gmi_selection_time, 475
dinfitem.mio_dual_bound_after_presolve, 475
dinfitem.mio_construct_solution_obj, 475
dinfitem.mio_cmir_separation_time, 475
dinfitem.mio_cmir_selection_time, 475
dinfitem.mio_clique_separation_time, 475
dinfitem.mio_clique_selection_time, 475
dinfitem.intpnt_time, 475
dinfitem.intpnt_primal_obj, 475
dinfitem.intpnt_primal_feas, 475
dinfitem.intpnt_order_time, 475
dinfitem.intpnt_opt_status, 475
dinfitem.intpnt_factor_num_flops, 474
dinfitem.intpnt_dual_obj, 474
dinfitem.intpnt_dual_feas, 474
dinfitem.folding_time, 474
dinfitem.folding_factor, 474
dinfitem.folding_bi_unfold_time, 474
dinfitem.folding_bi_unfold_primal_time, 474
dinfitem.folding_bi_unfold_initialize_time, 474
dinfitem.folding_bi_unfold_dual_time, 474
dinfitem.folding_bi_optimize_time, 474
dinfitem.bi_time, 474
dinfitem.bi_primal_time, 474
dinfitem.bi_dual_time, 474
dinfitem.bi_clean_time, 474
dinfitem.ana_pro_scalarized_constraint_matrix_density, 474
domaintype, 472
domaintype.svec_psd_cone, 473
domaintype.rzero, 472
domaintype.rquadratic_cone, 472
domaintype.rplus, 472
domaintype.rminus, 472
domaintype.r, 472
domaintype.quadratic_cone, 472
domaintype.primal_power_cone, 473
domaintype.primal_geo_mean_cone, 473
domaintype.primal_exp_cone, 473
domaintype.dual_power_cone, 473
domaintype.dual_geo_mean_cone, 473
domaintype.dual_exp_cone, 473
dparam, 369
feature, 480
feature.pts, 480
feature.pton, 480
foldingmode, 491
foldingmode.off, 491
foldingmode.free_unless_basic, 491
foldingmode.free, 491
foldingmode.force, 491
iinfitem, 481
iinfitem.sto_num_a_realloc, 488
iinfitem.sol_itr_solsta, 487
iinfitem.sol_itr_prosta, 487
iinfitem.sol_itg_solsta, 487
iinfitem.sol_itg_prosta, 487
iinfitem.sol_bas_solsta, 487
iinfitem.sol_bas_prosta, 487
iinfitem.sim_solve_dual, 487
iinfitem.sim_primal_iter, 487
iinfitem.sim_primal_inf_iter, 487
iinfitem.sim_primal_hotstart_lu, 487
iinfitem.sim_primal_hotstart, 487
iinfitem.sim_primal_deg_iter, 487
iinfitem.sim_numvar, 487
iinfitem.sim_numcon, 487
iinfitem.sim_dual_iter, 487
iinfitem.sim_dual_inf_iter, 487
iinfitem.sim_dual_hotstart_lu, 487
iinfitem.sim_dual_hotstart, 487
iinfitem.sim_dual_deg_iter, 487
iinfitem.rd_protype, 487
iinfitem.rd_numvar, 487
iinfitem.rd_numq, 487
iinfitem.rd_numintvar, 486
iinfitem.rd_numcone, 486
iinfitem.rd_numcon, 486
iinfitem.rd_numbarvar, 486
iinfitem.purify_primal_success, 486
iinfitem.purify_dual_success, 486
iinfitem.presolve_num_primal_perturbations, 486
iinfitem.optimize_response, 486
iinfitem.opt_numvar, 486
iinfitem.opt_numcon, 486
iinfitem.mio_user_obj_cut, 486
iinfitem.mio_total_num_separated_cuts, 486
iinfitem.mio_total_num_selected_cuts, 486

iinfitem.mio_relgap_satisfied, 486
iinfitem.mio_presolved_numvar, 486
iinfitem.mio_presolved_numrqcones, 486
iinfitem.mio_presolved_numqcones, 486
iinfitem.mio_presolved_numppowcones, 486
iinfitem.mio_presolved_numpexpcones, 486
iinfitem.mio_presolved_numintconevar, 486
iinfitem.mio_presolved_numint, 486
iinfitem.mio_presolved_numdpowcones, 486
iinfitem.mio_presolved_numdjc, 485
iinfitem.mio_presolved_numdexpcones, 485
iinfitem.mio_presolved_numcontconevar, 485
iinfitem.mio_presolved_numcont, 485
iinfitem.mio_presolved_numconevar, 485
iinfitem.mio_presolved_numcone, 485
iinfitem.mio_presolved_numcon, 485
iinfitem.mio_presolved_numbinconevar, 485
iinfitem.mio_presolved_numbin, 485
iinfitem.mio_obj_bound_defined, 485
iinfitem.mio_numvar, 485
iinfitem.mio_numrqcones, 485
iinfitem.mio_numqcones, 485
iinfitem.mio_numppowcones, 485
iinfitem.mio_numpexpcones, 485
iinfitem.mio_numintconevar, 485
iinfitem.mio_numint, 485
iinfitem.mio_numdpowcones, 485
iinfitem.mio_numdjc, 485
iinfitem.mio_numdexpcones, 485
iinfitem.mio_numcontconevar, 485
iinfitem.mio_numcont, 484
iinfitem.mio_numconevar, 484
iinfitem.mio_numcone, 484
iinfitem.mio_numcon, 484
iinfitem.mio_numbinconevar, 484
iinfitem.mio_numbin, 484
iinfitem.mio_num_solved_nodes, 484
iinfitem.mio_num_separated_lipro_cuts, 484
iinfitem.mio_num_separated_knapsack_cover_cuts, 484
iinfitem.mio_num_separated_implied_bound_cuts, 484
iinfitem.mio_num_separated_gomory_cuts, 484
iinfitem.mio_num_separated_cmir_cuts, 484
iinfitem.mio_num_separated_clique_cuts, 484
iinfitem.mio_num_selected_lipro_cuts, 484
iinfitem.mio_num_selected_knapsack_cover_cuts, 484
iinfitem.mio_num_selected_implied_bound_cuts, 484
iinfitem.mio_num_selected_gomory_cuts, 484
iinfitem.mio_num_selected_cmir_cuts, 484
iinfitem.mio_num_selected_clique_cuts, 484
iinfitem.mio_num_root_cut_rounds, 484
iinfitem.mio_num_restarts, 484
iinfitem.mio_num_repeated_presolve, 484
iinfitem.mio_num_relax, 483
iinfitem.mio_num_int_solutions, 483
iinfitem.mio_num_branch, 483
iinfitem.mio_num_blocks_solved_in_presolve, 483
iinfitem.mio_num_blocks_solved_in_bb, 483
iinfitem.mio_num_active_root_cuts, 483
iinfitem.mio_num_active_nodes, 483
iinfitem.mio_node_depth, 483
iinfitem.mio_initial_feasible_solution, 483
iinfitem.mio_final_numvar, 483
iinfitem.mio_final_numrqcones, 483
iinfitem.mio_final_numqcones, 483
iinfitem.mio_final_numppowcones, 483
iinfitem.mio_final_numpexpcones, 483
iinfitem.mio_final_numintconevar, 483
iinfitem.mio_final_numint, 483
iinfitem.mio_final_numdpowcones, 483
iinfitem.mio_final_numdjc, 483
iinfitem.mio_final_numdexpcones, 483
iinfitem.mio_final_numcontconevar, 483
iinfitem.mio_final_numcont, 483
iinfitem.mio_final_numconevar, 482
iinfitem.mio_final_numcone, 482
iinfitem.mio_final_numcon, 482
iinfitem.mio_final_numbinconevar, 482
iinfitem.mio_final_numbin, 482
iinfitem.mio_construct_solution, 482
iinfitem.mio_clique_table_size, 482
iinfitem.mio_absgap_satisfied, 482
iinfitem.intpnt_solve_dual, 482
iinfitem.intpnt_num_threads, 482
iinfitem.intpnt_iter, 482
iinfitem.intpnt_factor_dim_dense, 482
iinfitem.folding_applied, 482
iinfitem.ana_pro_num_var_up, 482
iinfitem.ana_pro_num_var_ra, 482
iinfitem.ana_pro_num_var_lo, 482
iinfitem.ana_pro_num_var_int, 482
iinfitem.ana_pro_num_var_fr, 482
iinfitem.ana_pro_num_var_eq, 482
iinfitem.ana_pro_num_var_cont, 482
iinfitem.ana_pro_num_var_bin, 481
iinfitem.ana_pro_num_var, 481
iinfitem.ana_pro_num_con_up, 481
iinfitem.ana_pro_num_con_ra, 481
iinfitem.ana_pro_num_con_lo, 481
iinfitem.ana_pro_num_con_fr, 481
iinfitem.ana_pro_num_con_eq, 481
iinfitem.ana_pro_num_con, 481
inftype, 488
inftype.lint_type, 488
inftype.int_type, 488
inftype.dou_type, 488
intpnthotstart, 466
intpnthotstart.primal_dual, 466
intpnthotstart.primal, 466
intpnthotstart.none, 466
intpnthotstart.dual, 466
iomode, 488

iomode.write, 488
iomode.readwrite, 488
iomode.read, 488
iparam, 386
liinfitem, 480
liinfitem.simplex_iter, 481
liinfitem.rd_numqnz, 481
liinfitem.rd_numdjc, 481
liinfitem.rd_numanz, 481
liinfitem.rd_numacc, 481
liinfitem.mio_simplex_iter, 481
liinfitem.mio_presolved_anz, 481
liinfitem.mio_num_prim_illposed_cer, 481
liinfitem.mio_num_dual_illposed_cer, 481
liinfitem.mio_intpnt_iter, 481
liinfitem.mio_final_anz, 481
liinfitem.mio_anz, 480
liinfitem.intpnt_factor_num_nz, 480
liinfitem.folding_bi_primal_iter, 480
liinfitem.folding_bi_optimizer_iter, 480
liinfitem.folding_bi_dual_iter, 480
liinfitem.bi_primal_iter, 480
liinfitem.bi_dual_iter, 480
liinfitem.bi_clean_iter, 480
liinfitem.ana_pro_scalarized_constraint_matrix_num_nz, 480
liinfitem.ana_pro_scalarized_constraint_matrix_num_var, 480
liinfitem.ana_pro_scalarized_constraint_matrix_num_con, 480
mark, 465
mark.up, 465
mark.lo, 465
miocontsoltype, 489
miocontsoltype.root, 489
miocontsoltype.none, 489
miocontsoltype.itg_rel, 489
miocontsoltype.itg, 489
miodatapermmethod, 489
miodatapermmethod.random, 489
miodatapermmethod.none, 489
miodatapermmethod.cyclic_shift, 489
miomode, 489
miomode.satisfied, 489
miomode.ignored, 489
mionodeseltype, 489
mionodeseltype.pseudo, 489
mionodeseltype.free, 489
mionodeseltype.first, 489
mionodeseltype.best, 489
miovarseltype, 489
miovarseltype.strong, 490
miovarseltype.pseudocost, 490
miovarseltype.free, 490
miqcqoreformmethod, 488
miqcqoreformmethod.relax_sdp, 489
miqcqoreformmethod.none, 488
miqcqoreformmethod.linearization, 488
miqcqoreformmethod.free, 488
miqcqoreformmethod.eigen_val_method, 489
miqcqoreformmethod.diag_sdp, 489
mpsformat, 490
mpsformat.strict, 490
mpsformat.relaxed, 490
mpsformat.free, 490
mpsformat.cplex, 490
nametype, 473
nametype.mps, 473
nametype.lp, 473
nametype.gen, 473
objsense, 490
objsense.minimize, 490
objsense.maximize, 490
onoffkey, 490
onoffkey.on, 490
onoffkey.off, 490
optimizertype, 490
optimizertype.primal_simplex, 491
optimizertype.new_primal_simplex, 490
optimizertype.new_dual_simplex, 490
optimizertype.mixed_int, 490
optimizertype.intpnt, 490
optimizertype.free_simplex, 490
optimizertype.free, 490
optimizertype.dual_simplex, 490
optimizertype.conic, 490
orderingtype, 491
orderingtype.try_graphpar, 491
orderingtype.none, 491
orderingtype.free, 491
orderingtype.force_graphpar, 491
orderingtype.experimental, 491
orderingtype.appminloc, 491
parametertype, 491
parametertype.str_type, 491
parametertype.invalid_type, 491
parametertype.int_type, 491
parametertype.dou_type, 491
presolvemode, 491
presolvemode.on, 491
presolvemode.off, 491
presolvemode.free, 491
problemitem, 492
problemitem.var, 492
problemitem.cone, 492
problemitem.con, 492
problemtype, 492
problemtype.qo, 492
problemtype.qcqo, 492
problemtype.mixed, 492
problemtype.lo, 492
problemtype.conic, 492
prosta, 492
prosta.unknown, 492
prosta.prim_infeas_or_unbounded, 492
prosta.prim_infeas, 492

prosta.prim_feas, 492
prosta.prim_and_dual_infeas, 492
prosta.prim_and_dual_feas, 492
prosta.ill_posed, 492
prosta.dual_infeas, 492
prosta.dual_feas, 492
rescode, 441
rescodetype, 492
rescodetype.wrn, 493
rescodetype.unk, 493
rescodetype.trm, 493
rescodetype.ok, 492
rescodetype.err, 493
scalingmethod, 493
scalingmethod.pow2, 493
scalingmethod.free, 493
scalingtype, 493
scalingtype.none, 493
scalingtype.free, 493
sensitivitytype, 493
sensitivitytype.basis, 493
simdegen, 465
simdegen.none, 465
simdegen.moderate, 465
simdegen.minimum, 465
simdegen.free, 465
simdegen.aggressive, 465
simdupvec, 466
simdupvec.on, 466
simdupvec.off, 466
simdupvec.free, 466
simhotstart, 466
simhotstart.status_keys, 466
simhotstart.none, 466
simhotstart.free, 466
simprecision, 465
simprecision.normal, 465
simprecision.extended, 465
simreform, 466
simreform.on, 466
simreform.off, 466
simreform.free, 466
simreform.aggressive, 466
simseltype, 493
simseltype.se, 493
simseltype.partial, 493
simseltype.full, 493
simseltype.free, 493
simseltype.devex, 493
simseltype.ase, 493
solformat, 474
solformat.task, 474
solformat.json_task, 474
solformat.extension, 474
solformat.b, 474
solitem, 493
solitem.y, 494
solitem.xx, 493
solitem.xc, 493
solitem.sux, 494
solitem.suc, 494
solitem.snx, 494
solitem.slx, 494
solitem.slc, 494
solsta, 494
solsta.unknown, 494
solsta.prim_infeas_cer, 494
solsta.prim_illposed_cer, 494
solsta.prim_feas, 494
solsta.prim_and_dual_feas, 494
solsta.optimal, 494
solsta.integer_optimal, 494
solsta.dual_infeas_cer, 494
solsta.dual_illposed_cer, 494
solsta.dual_feas, 494
soltype, 494
soltype.itr, 494
soltype.itg, 494
soltype.bas, 494
solveform, 494
solveform.primal, 495
solveform.free, 494
solveform.dual, 495
sparam, 436
stakey, 495
stakey.upr, 495
stakey.unk, 495
stakey.supbas, 495
stakey.low, 495
stakey.inf, 495
stakey.fix, 495
stakey.bas, 495
startpointtype, 495
startpointtype.guess, 495
startpointtype.free, 495
startpointtype.constant, 495
streamtype, 495
streamtype.wrn, 495
streamtype.msg, 495
streamtype.log, 495
streamtype.err, 495
symmattype, 473
symmattype.sparse, 473
transpose, 465
transpose.yes, 465
transpose.no, 465
uplo, 465
uplo.up, 466
uplo.lo, 465
value, 495
value.max_str_len, 495
value.license_buffer_length, 495
variabletype, 496
variabletype.type_int, 496
variabletype.type_cont, 496

Exceptions

Error, 356
MosekException, 356

Parameters

Double parameters, 369
dparam.ana_sol_infeas_tol, 369
dparam.basis_rel_tol_s, 369
dparam.basis_tol_s, 369
dparam.basis_tol_x, 369
dparam.data_sym_mat_tol, 370
dparam.data_sym_mat_tol_huge, 370
dparam.data_sym_mat_tol_large, 370
dparam.data_tol_aij_huge, 370
dparam.data_tol_aij_large, 371
dparam.data_tol_bound_inf, 371
dparam.data_tol_bound_wrn, 371
dparam.data_tol_c_huge, 371
dparam.data_tol_cj_large, 372
dparam.data_tol_qij, 372
dparam.data_tol_x, 372
dparam.folding_tol_eq, 372
dparam.intpnt_co_tol_dfeas, 373
dparam.intpnt_co_tol_infeas, 373
dparam.intpnt_co_tol_mu_red, 373
dparam.intpnt_co_tol_near_rel, 374
dparam.intpnt_co_tol_pfeas, 374
dparam.intpnt_co_tol_rel_gap, 374
dparam.intpnt_qo_tol_dfeas, 374
dparam.intpnt_qo_tol_infeas, 375
dparam.intpnt_qo_tol_mu_red, 375
dparam.intpnt_qo_tol_near_rel, 375
dparam.intpnt_qo_tol_pfeas, 375
dparam.intpnt_qo_tol_rel_gap, 376
dparam.intpnt_tol_dfeas, 376
dparam.intpnt_tol_dsafe, 376
dparam.intpnt_tol_infeas, 377
dparam.intpnt_tol_mu_red, 377
dparam.intpnt_tol_path, 377
dparam.intpnt_tol_pfeas, 377
dparam.intpnt_tol_psafe, 378
dparam.intpnt_tol_rel_gap, 378
dparam.intpnt_tol_rel_step, 378
dparam.intpnt_tol_step_size, 378
dparam.lower_obj_cut, 379
dparam.lower_obj_cut_finite_trh, 379
dparam.mio_clique_table_size_factor, 379
dparam.mio_djc_max_bigm, 379
dparam.mio_max_time, 380
dparam.mio_rel_gap_const, 380
dparam.mio_tol_abs_gap, 380
dparam.mio_tol_abs_relax_int, 381
dparam.mio_tol_feas, 381
dparam.mio_tol_rel_dual_bound_improvement, 381
dparam.mio_tol_rel_gap, 381
dparam.optimizer_max_ticks, 382
dparam.optimizer_max_time, 382
dparam.presolve_tol_abs_lindep, 382
dparam.presolve_tol_primal_infeas_perturbation, 382
dparam.presolve_tol_rel_lindep, 383
dparam.presolve_tol_s, 383
dparam.presolve_tol_x, 383
dparam.qcqo_reformulate_rel_drop_tol, 383
dparam.semidefinite_tol_approx, 384
dparam.sim_lu_tol_rel_piv, 384
dparam.sim_precision_scaling_extended, 384
dparam.sim_precision_scaling_normal, 384
dparam.simplex_abs_tol_piv, 385
dparam.upper_obj_cut, 385
dparam.upper_obj_cut_finite_trh, 385
Integer parameters, 386
iparam.ana_sol_basis, 386
iparam.ana_sol_print_violated, 386
iparam.auto_sort_a_before_opt, 386
iparam.auto_update_sol_info, 386
iparam.basis_solve_use_plus_one, 387
iparam.bi_clean_optimizer, 387
iparam.bi_ignore_max_iter, 387
iparam.bi_ignore_num_error, 387
iparam.bi_max_iterations, 388
iparam.cache_license, 388
iparam.compress_statfile, 388
iparam.folding_use, 389
iparam.getdual_convert_lmis, 389
iparam.heartbeat_sim_freq_ticks, 389
iparam.infeas_generic_names, 389
iparam.infeas_report_auto, 390
iparam.infeas_report_level, 390
iparam.intpnt_basis, 390
iparam.intpnt_diff_step, 390
iparam.intpnt_hotstart, 391
iparam.intpnt_max_iterations, 391
iparam.intpnt_max_num_cor, 391
iparam.intpnt_off_col_trh, 391
iparam.intpnt_order_gp_num_seeds, 392
iparam.intpnt_order_method, 392
iparam.intpnt_regularization_use, 392
iparam.intpnt_scaling, 393
iparam.intpnt_solve_form, 393
iparam.intpnt_starting_point, 393
iparam.license_debug, 393
iparam.license_pause_time, 394
iparam.license_suppress_expire_wrns, 394
iparam.license_trh_expiry_wrn, 394
iparam.license_wait, 394
iparam.log, 395
iparam.log_ana_pro, 395
iparam.log_bi, 395
iparam.log_bi_freq, 395
iparam.log_cut_second_opt, 396
iparam.log_expand, 396
iparam.log_feas_repair, 396
iparam.log_file, 397
iparam.log_include_summary, 397

iparam.log_infeas_ana, 397
iparam.log_intpnt, 397
iparam.log_local_info, 398
iparam.log_mio, 398
iparam.log_mio_freq, 398
iparam.log_order, 398
iparam.log_presolve, 399
iparam.log_sensitivity, 399
iparam.log_sensitivity_opt, 399
iparam.log_sim, 399
iparam.log_sim_freq, 400
iparam.log_sim_freq_giga_ticks, 400
iparam.log_storage, 400
iparam.max_num_warnings, 401
iparam.mio_branch_dir, 401
iparam.mio_conflict_analysis_level, 401
iparam.mio_conic_outer_approximation, 401
iparam.mio_construct_sol, 402
iparam.mio_crossover_max_nodes, 402
iparam.mio_cut_clique, 402
iparam.mio_cut_cmir, 402
iparam.mio_cut_gmi, 403
iparam.mio_cut_implied_bound, 403
iparam.mio_cut_knapsack_cover, 403
iparam.mio_cut_lipro, 403
iparam.mio_cut_selection_level, 404
iparam.mio_data_permutation_method, 404
iparam.mio_dual_ray_analysis_level, 404
iparam.mio_feaspump_level, 405
iparam.mio_heuristic_level, 405
iparam.mio_independent_block_level, 405
iparam.mio_max_num_branches, 406
iparam.mio_max_num_relaxs, 406
iparam.mio_max_num_restarts, 406
iparam.mio_max_num_root_cut_rounds, 406
iparam.mio_max_num_solutions, 407
iparam.mio_memory_emphasis_level, 407
iparam.mio_min_rel, 407
iparam.mio_mode, 408
iparam.mio_node_optimizer, 408
iparam.mio_node_selection, 408
iparam.mio_numerical_emphasis_level, 408
iparam.mio_opt_face_max_nodes, 409
iparam.mio_perspective_reformulate, 409
iparam.mio_presolve_aggregator_use, 409
iparam.mio_probing_level, 409
iparam.mio_propagate_objective_constraint, 410
iparam.mio_qcqo_reformulation_method, 410
iparam.mio_rens_max_nodes, 410
iparam.mio_rins_max_nodes, 411
iparam.mio_root_optimizer, 411
iparam.mio_seed, 411
iparam.mio_symmetry_level, 411
iparam.mio_var_selection, 412
iparam.mio_vb_detection_level, 412
iparam.mt_spincount, 412
iparam.ng, 413
iparam.num_threads, 413
iparam.opf_write_header, 413
iparam.opf_write_hints, 413
iparam.opf_write_line_length, 414
iparam.opf_write_parameters, 414
iparam.opf_write_problem, 414
iparam.opf_write_sol_bas, 414
iparam.opf_write_sol_itg, 415
iparam.opf_write_sol_itr, 415
iparam.opf_write_solutions, 415
iparam.optimizer, 415
iparam.param_read_case_name, 416
iparam.param_read_ign_error, 416
iparam.presolve_eliminator_max_fill, 416
iparam.presolve_eliminator_max_num_tries, 416
iparam.presolve_lindep_abs_work_trh, 417
iparam.presolve_lindep_new, 417
iparam.presolve_lindep_rel_work_trh, 417
iparam.presolve_lindep_use, 417
iparam.presolve_max_num_pass, 418
iparam.presolve_max_num_reductions, 418
iparam.presolve_use, 418
iparam.primal_repair_optimizer, 418
iparam.ptf_write_parameters, 419
iparam.ptf_write_single_psd_terms, 419
iparam.ptf_write_solutions, 419
iparam.ptf_write_transform, 419
iparam.read_async, 420
iparam.read_debug, 420
iparam.read_keep_free_con, 420
iparam.read_mps_format, 420
iparam.read_mps_width, 421
iparam.read_task_ignore_param, 421
iparam.remote_use_compression, 421
iparam.remove_unused_solutions, 421
iparam.sensitivity_all, 422
iparam.sensitivity_type, 422
iparam.sim_basis_factor_use, 422
iparam.sim_degen, 422
iparam.sim_detect_pwl, 423
iparam.sim_dual_crash, 423
iparam.sim_dual_phaseone_method, 423
iparam.sim_dual_restrict_selection, 423
iparam.sim_dual_selection, 424
iparam.sim_exploit_dupvec, 424
iparam.sim_hotstart, 424
iparam.sim_hotstart_lu, 425
iparam.sim_max_iterations, 425
iparam.sim_max_num_setbacks, 425
iparam.sim_non_singular, 425
iparam.sim_precision, 426
iparam.sim_precision_boost, 426
iparam.sim_primal_crash, 426
iparam.sim_primal_phaseone_method, 426
iparam.sim_primal_restrict_selection, 427
iparam.sim_primal_selection, 427
iparam.sim_refactor_freq, 427

iparam.sim_reformulation, 427
iparam.sim_save_lu, 428
iparam.sim_scaling, 428
iparam.sim_scaling_method, 428
iparam.sim_seed, 428
iparam.sim_solve_form, 429
iparam.sim_switch_optimizer, 429
iparam.sol_filter_keep_basic, 429
iparam.sol_read_name_width, 430
iparam.sol_read_width, 430
iparam.timing_level, 430
iparam.write_async, 430
iparam.write_bas_constraints, 431
iparam.write_bas_head, 431
iparam.write_bas_variables, 431
iparam.write_compression, 431
iparam.write_free_con, 432
iparam.write_generic_names, 432
iparam.write_ignore_incompatible_items, 432
iparam.write_int_constraints, 432
iparam.write_int_head, 433
iparam.write_int_variables, 433
iparam.write_json_indentation, 433
iparam.write_lp_full_obj, 433
iparam.write_lp_line_width, 434
iparam.write_mps_format, 434
iparam.write_mps_int, 434
iparam.write_sol_barvariables, 434
iparam.write_sol_constraints, 435
iparam.write_sol_head, 435
iparam.write_sol_ignore_invalid_names, 435
iparam.write_sol_variables, 435
String parameters, 436
sparam.bas_sol_file_name, 436
sparam.data_file_name, 436
sparam.debug_file_name, 436
sparam.int_sol_file_name, 436
sparam.itr_sol_file_name, 436
sparam.mio_debug_string, 437
sparam.param_comment_sign, 437
sparam.param_read_file_name, 437
sparam.param_write_file_name, 437
sparam.read_mps_bou_name, 437
sparam.read_mps_obj_name, 438
sparam.read_mps_ran_name, 438
sparam.read_mps_rhs_name, 438
sparam.remote_optserver_host, 438
sparam.remote_tls_cert, 439
sparam.remote_tls_cert_path, 439
sparam.sensitivity_file_name, 439
sparam.sensitivity_res_file_name, 439
sparam.sol_filter_xc_low, 439
sparam.sol_filter_xc_upr, 440
sparam.sol_filter_xx_low, 440
sparam.sol_filter_xx_upr, 440
sparam.stat_key, 440
sparam.stat_name, 441

Response codes

Termination, 441
rescode.ok, 441
rescode.trm_internal, 442
rescode.trm_internal_stop, 442
rescode.trm_lost_race, 442
rescode.trm_max_iterations, 441
rescode.trm_max_num_setbacks, 442
rescode.trm_max_time, 441
rescode.trm_mio_num_branches, 441
rescode.trm_mio_num_relaxs, 441
rescode.trm_num_max_num_int_solutions, 441
rescode.trm_numerical_problem, 442
rescode.trm_objective_range, 441
rescode.trm_server_max_memory, 442
rescode.trm_server_max_time, 442
rescode.trm_stall, 441
rescode.trm_user_callback, 442
Warnings, 442
rescode.wrn_ana_almost_int_bounds, 444
rescode.wrn_ana_c_zero, 444
rescode.wrn_ana_close_bounds, 444
rescode.wrn_ana_empty_cols, 444
rescode.wrn_ana_large_bounds, 444
rescode.wrn_dropped_nz_qobj, 443
rescode.wrn_duplicate_barvariable_names, 444
rescode.wrn_duplicate_cone_names, 444
rescode.wrn_duplicate_constraint_names, 444
rescode.wrn_duplicate_variable_names, 444
rescode.wrn_eliminator_space, 444
rescode.wrn_empty_name, 443
rescode.wrn_getdual_ignores_integrality, 444
rescode.wrn_ignore_integer, 443
rescode.wrn_incomplete_linear_dependency_check, 444
rescode.wrn_invalid_mps_name, 443
rescode.wrn_invalid_mps_obj_name, 443
rescode.wrn_large_aij, 442
rescode.wrn_large_bound, 442
rescode.wrn_large_cj, 442
rescode.wrn_large_con_fx, 442
rescode.wrn_large_fij, 445
rescode.wrn_large_lo_bound, 442
rescode.wrn_large_up_bound, 442
rescode.wrn_license_expire, 443
rescode.wrn_license_feature_expire, 443
rescode.wrn_license_server, 443
rescode.wrn_lp_drop_variable, 443
rescode.wrn_lp_old_quad_format, 442
rescode.wrn_mio_infeasible_final, 443
rescode.wrn_modified_double_parameter, 445
rescode.wrn_mps_split_bou_vector, 442
rescode.wrn_mps_split_ran_vector, 442
rescode.wrn_mps_split_rhs_vector, 442
rescode.wrn_name_max_len, 442
rescode.wrn_no_dualizer, 444

rescode.wrn_no_global_optimizer, 443
rescode.wrn_no_infeasibility_report_when_matrix_variables, 444
rescode.wrn_nz_in_upr_tri, 443
rescode.wrn_open_param_file, 442
rescode.wrn_param_ignored_cmio, 443
rescode.wrn_param_name_dou, 443
rescode.wrn_param_name_int, 443
rescode.wrn_param_name_str, 443
rescode.wrn_param_str_value, 443
rescode.wrn_presolve_outofspace, 444
rescode.wrn_presolve_primal_perturbations, 444
rescode.wrn_ptf_unknown_section, 445
rescode.wrn_sol_file_ignored_con, 443
rescode.wrn_sol_file_ignored_var, 443
rescode.wrn_sol_filter, 443
rescode.wrn_spar_max_len, 442
rescode.wrn_sym_mat_large, 444
rescode.wrn_too_few_basis_vars, 443
rescode.wrn_too_many_basis_vars, 443
rescode.wrn_undef_sol_file_name, 443
rescode.wrn_using_generic_names, 443
rescode.wrn_write_changed_names, 444
rescode.wrn_write_discarded_cfix, 444
rescode.wrn_zero_aij, 442
rescode.wrn_zeros_in_sparse_col, 444
rescode.wrn_zeros_in_sparse_row, 444
Errors, 445
rescode.err_acc_afe_domain_mismatch, 464
rescode.err_acc_invalid_entry_index, 464
rescode.err_acc_invalid_index, 464
rescode.err_ad_invalid_codelist, 458
rescode.err_afe_invalid_index, 464
rescode.err_api_array_too_small, 458
rescode.err_api_cb_connect, 458
rescode.err_api_fatal_error, 458
rescode.err_api_internal, 458
rescode.err_appending_too_big_cone, 454
rescode.err_arg_is_too_large, 451
rescode.err_arg_is_too_small, 451
rescode.err_argument_dimension, 451
rescode.err_argument_is_too_large, 459
rescode.err_argument_is_too_small, 459
rescode.err_argument_lenneq, 450
rescode.err_argument_perm_array, 454
rescode.err_argument_type, 451
rescode.err_axis_name_specification, 447
rescode.err_bar_var_dim, 458
rescode.err_basis, 453
rescode.err_basis_factor, 456
rescode.err_basis_singular, 456
rescode.err_blank_name, 447
rescode.err_cbf_duplicate_acoord, 461
rescode.err_cbf_duplicate_bcoord, 461
rescode.err_cbf_duplicate_con, 460
rescode.err_cbf_duplicate_int, 460
rescode.err_cbf_duplicate_obj, 460
rescode.err_cbf_duplicate_objacoord, 461
rescode.err_cbf_duplicate_pow_cones, 461
rescode.err_cbf_duplicate_pow_star_cones, 461
rescode.err_cbf_duplicate_psdcon, 462
rescode.err_cbf_duplicate_psdvar, 461
rescode.err_cbf_duplicate_var, 460
rescode.err_cbf_expected_a_keyword, 462
rescode.err_cbf_invalid_con_type, 461
rescode.err_cbf_invalid_dimension_of_cones, 462
rescode.err_cbf_invalid_dimension_of_psdcon, 462
rescode.err_cbf_invalid_domain_dimension, 461
rescode.err_cbf_invalid_exp_dimension, 461
rescode.err_cbf_invalid_int_index, 461
rescode.err_cbf_invalid_num_acoord, 462
rescode.err_cbf_invalid_num_bcoord, 462
rescode.err_cbf_invalid_num_dcoord, 462
rescode.err_cbf_invalid_num_fcoord, 462
rescode.err_cbf_invalid_num_hcoord, 462
rescode.err_cbf_invalid_num_objacoord, 462
rescode.err_cbf_invalid_num_objfcoord, 462
rescode.err_cbf_invalid_num_psdcon, 462
rescode.err_cbf_invalid_number_of_cones, 462
rescode.err_cbf_invalid_power, 461
rescode.err_cbf_invalid_power_cone_index, 461
rescode.err_cbf_invalid_power_star_cone_index, 461
rescode.err_cbf_invalid_psdcon_block_index, 462
rescode.err_cbf_invalid_psdcon_index, 462
rescode.err_cbf_invalid_psdcon_variable_index, 462
rescode.err_cbf_invalid_psdvar_dimension, 461
rescode.err_cbf_invalid_var_type, 461
rescode.err_cbf_no_variables, 460
rescode.err_cbf_no_version_specified, 460
rescode.err_cbf_obj_sense, 460
rescode.err_cbf_parse, 460
rescode.err_cbf_power_cone_is_too_long, 461
rescode.err_cbf_power_cone_mismatch, 461
rescode.err_cbf_power_star_cone_mismatch, 461
rescode.err_cbf_syntax, 460
rescode.err_cbf_too_few_constraints, 461
rescode.err_cbf_too_few_ints, 461
rescode.err_cbf_too_few_psdvar, 461
rescode.err_cbf_too_few_variables, 461
rescode.err_cbf_too_many_constraints, 460
rescode.err_cbf_too_many_ints, 461
rescode.err_cbf_too_many_variables, 460
rescode.err_cbf_unhandled_power_cone_type, 461

rescode.err_cbf_unhandled_power_star_cone_type, 461
rescode.err_cbf_unsupported, 461
rescode.err_cbf_unsupported_change, 462
rescode.err_con_q_not_nsd, 454
rescode.err_con_q_not_psd, 454
rescode.err_cone_index, 454
rescode.err_cone_overlap, 454
rescode.err_cone_overlap_append, 454
rescode.err_cone_parameter, 454
rescode.err_cone_rep_var, 454
rescode.err_cone_size, 454
rescode.err_cone_type, 454
rescode.err_cone_type_str, 454
rescode.err_data_file_ext, 446
rescode.err_dimension_specification, 447
rescode.err_djc_afe_domain_mismatch, 464
rescode.err_djc_domain_termsize_mismatch, 464
rescode.err_djc_invalid_index, 464
rescode.err_djc_invalid_term_size, 464
rescode.err_djc_total_num_terms_mismatch, 464
rescode.err_djc_unsupported_domain_type, 464
rescode.err_domain_dimension, 463
rescode.err_domain_dimension_psd, 463
rescode.err_domain_invalid_index, 463
rescode.err_domain_power_invalid_alpha, 463
rescode.err_domain_power_negative_alpha, 463
rescode.err_domain_power_nleft, 463
rescode.err_dup_name, 447
rescode.err_duplicate_aij, 454
rescode.err_duplicate_barvariable_names, 459
rescode.err_duplicate_cone_names, 459
rescode.err_duplicate_constraint_names, 459
rescode.err_duplicate_djc_names, 459
rescode.err_duplicate_domain_names, 459
rescode.err_duplicate_fij, 463
rescode.err_duplicate_index_in_a_sparse_matrix, 463
rescode.err_duplicate_index_in_afeidx_list, 463
rescode.err_duplicate_variable_names, 459
rescode.err_end_of_file, 447
rescode.err_factor, 456
rescode.err_feasrepair_cannot_relax, 456
rescode.err_feasrepair_inconsistent_bound, 457
rescode.err_feasrepair_solving_relaxed, 457
rescode.err_file_license, 445
rescode.err_file_open, 446
rescode.err_file_read, 446
rescode.err_file_write, 446
rescode.err_final_solution, 456
rescode.err_first, 456
rescode.err_firsti, 453
rescode.err_firstj, 453
rescode.err_fixed_bound_values, 455
rescode.err_flexlm, 445
rescode.err_format_string, 447
rescode.err_getdual_not_available, 462
rescode.err_global_inv_conic_problem, 456
rescode.err_huge_aij, 454
rescode.err_huge_c, 454
rescode.err_huge_fij, 463
rescode.err_identical_tasks, 458
rescode.err_in_argument, 451
rescode.err_index, 452
rescode.err_index_arr_is_too_large, 451
rescode.err_index_arr_is_too_small, 451
rescode.err_index_is_not_unique, 451
rescode.err_index_is_too_large, 451
rescode.err_index_is_too_small, 451
rescode.err_inf_dou_index, 451
rescode.err_inf_dou_name, 451
rescode.err_inf_in_double_data, 455
rescode.err_inf_int_index, 451
rescode.err_inf_int_name, 452
rescode.err_inf_lint_index, 451
rescode.err_inf_lint_name, 452
rescode.err_inf_type, 452
rescode.err_infeas_undefined, 458
rescode.err_infinite_bound, 455
rescode.err_int64_to_int32_cast, 458
rescode.err_internal, 458
rescode.err_internal_test_failed, 458
rescode.err_inv_aptre, 452
rescode.err_inv_bk, 452
rescode.err_inv_bkc, 452
rescode.err_inv_bkx, 452
rescode.err_inv_cone_type, 453
rescode.err_inv_cone_type_str, 453
rescode.err_inv_dinf, 453
rescode.err_inv_iinf, 453
rescode.err_inv_liinf, 453
rescode.err_inv_marki, 457
rescode.err_inv_markj, 457
rescode.err_inv_name_item, 453
rescode.err_inv_numi, 457
rescode.err_inv_numj, 457
rescode.err_inv_optimizer, 456
rescode.err_inv_problem, 456
rescode.err_inv_qcon_subi, 455
rescode.err_inv_qcon_subj, 455
rescode.err_inv_qcon_subk, 455
rescode.err_inv_qcon_val, 455
rescode.err_inv_qobj_subi, 455
rescode.err_inv_qobj_subj, 455
rescode.err_inv_qobj_val, 455
rescode.err_inv_rescode, 453
rescode.err_inv_sk, 453
rescode.err_inv_sk_str, 453
rescode.err_inv_skc, 453

rescode.err_inv_skn, 453
rescode.err_inv_skx, 453
rescode.err_inv_var_type, 452
rescode.err_invalid_aij, 456
rescode.err_invalid_b, 463
rescode.err_invalid_barvar_name, 447
rescode.err_invalid_cfix, 455
rescode.err_invalid_cj, 456
rescode.err_invalid_compression, 457
rescode.err_invalid_con_name, 447
rescode.err_invalid_cone_name, 447
rescode.err_invalid_fij, 463
rescode.err_invalid_file_format_for_affine_conic_constraints, 459
rescode.err_invalid_file_format_for_cfix, 459
rescode.err_invalid_file_format_for_cones, 459
rescode.err_invalid_file_format_for_disjunctive_constraints, 459
rescode.err_invalid_file_format_for_free_constraints, 459
rescode.err_invalid_file_format_for_nonlinear, 459
rescode.err_invalid_file_format_for_quadratic_terms, 459
rescode.err_invalid_file_format_for_ranged_constraints, 459
rescode.err_invalid_file_format_for_sym_mat, 459
rescode.err_invalid_file_name, 446
rescode.err_invalid_format_type, 453
rescode.err_invalid_g, 463
rescode.err_invalid_idx, 452
rescode.err_invalid_iomode, 457
rescode.err_invalid_max_num, 452
rescode.err_invalid_name_in_sol_file, 450
rescode.err_invalid_obj_name, 447
rescode.err_invalid_objective_sense, 455
rescode.err_invalid_problem_type, 459
rescode.err_invalid_sol_file_name, 446
rescode.err_invalid_stream, 447
rescode.err_invalid_surplus, 453
rescode.err_invalid_sym_mat_dim, 459
rescode.err_invalid_task, 447
rescode.err_invalid_utf8, 457
rescode.err_invalid_var_name, 447
rescode.err_invalid_wchar, 457
rescode.err_invalid_whichsol, 451
rescode.err_json_data, 450
rescode.err_json_format, 450
rescode.err_json_missing_data, 450
rescode.err_json_number_overflow, 450
rescode.err_json_string, 450
rescode.err_json_syntax, 450
rescode.err_last, 456
rescode.err_lasti, 453
rescode.err_lastj, 453
rescode.err_lau_arg_k, 460
rescode.err_lau_arg_m, 460
rescode.err_lau_arg_n, 460
rescode.err_lau_arg_trans, 460
rescode.err_lau_arg_transa, 460
rescode.err_lau_arg_transb, 460
rescode.err_lau_arg_uplo, 460
rescode.err_lau_invalid_lower_triangular_matrix, 460
rescode.err_lau_invalid_sparse_symmetric_matrix, 460
rescode.err_lau_not_positive_definite, 460
rescode.err_lau_singular_matrix, 460
rescode.err_lau_unknown, 460
rescode.err_license, 445
rescode.err_license_cannot_allocate, 445
rescode.err_license_cannot_connect, 446
rescode.err_license_expired, 445
rescode.err_license_feature, 445
rescode.err_license_invalid_hostid, 446
rescode.err_license_max, 445
rescode.err_license_moseklm_daemon, 445
rescode.err_license_no_server_line, 446
rescode.err_license_no_server_support, 446
rescode.err_license_old_server_version, 445
rescode.err_license_server, 445
rescode.err_license_server_version, 446
rescode.err_license_version, 445
rescode.err_link_file_dll, 446
rescode.err_living_tasks, 447
rescode.err_lower_bound_is_a_nan, 454
rescode.err_lp_ambiguous_constraint_bound, 450
rescode.err_lp_duplicate_section, 450
rescode.err_lp_empty, 449
rescode.err_lp_expected_constraint_relation, 450
rescode.err_lp_expected_number, 450
rescode.err_lp_expected_objective, 450
rescode.err_lp_file_format, 450
rescode.err_lp_indicator_var, 450
rescode.err_lp_invalid_var_name, 450
rescode.err_lu_max_num_tries, 457
rescode.err_max_len_is_too_small, 453
rescode.err_maxnumbarvar, 452
rescode.err_maxnumcon, 452
rescode.err_maxnumcone, 454
rescode.err_maxnumqnz, 452
rescode.err_maxnumvar, 452
rescode.err_mio_internal, 459
rescode.err_mio_invalid_node_optimizer, 462
rescode.err_mio_invalid_root_optimizer, 462
rescode.err_mio_no_optimizer, 456
rescode.err_mismatching_dimension, 447
rescode.err_missing_license_file, 445
rescode.err_mixed_conic_and_nl, 456
rescode.err_mps_cone_overlap, 448
rescode.err_mps_cone_repeat, 449

rescode.err_mps_cone_type, 448
rescode.err_mps_duplicate_q_element, 449
rescode.err_mps_file, 448
rescode.err_mps_inv_field, 448
rescode.err_mps_inv_marker, 448
rescode.err_mps_inv_sec_order, 448
rescode.err_mps_invalid_bound_key, 448
rescode.err_mps_invalid_con_key, 448
rescode.err_mps_invalid_indicator_constraint, 449
rescode.err_mps_invalid_indicator_quadratic_constraint, 449
rescode.err_mps_invalid_indicator_value, 449
rescode.err_mps_invalid_indicator_variable, 449
rescode.err_mps_invalid_key, 449
rescode.err_mps_invalid_obj_name, 449
rescode.err_mps_invalid_objsense, 449
rescode.err_mps_invalid_sec_name, 448
rescode.err_mps_mul_con_name, 448
rescode.err_mps_mul_csec, 448
rescode.err_mps_mul_qobj, 448
rescode.err_mps_mul_qsec, 448
rescode.err_mps_no_objective, 448
rescode.err_mps_non_symmetric_q, 449
rescode.err_mps_null_con_name, 448
rescode.err_mps_null_var_name, 448
rescode.err_mps_splitted_var, 448
rescode.err_mps_tab_in_field2, 449
rescode.err_mps_tab_in_field3, 449
rescode.err_mps_tab_in_field5, 449
rescode.err_mps_undef_con_name, 448
rescode.err_mps_undef_var_name, 448
rescode.err_mps_write_cplex_invalid_cone_type, 462
rescode.err_mul_a_element, 452
rescode.err_name_is_null, 457
rescode.err_name_max_len, 457
rescode.err_nan_in_blc, 455
rescode.err_nan_in_blx, 456
rescode.err_nan_in_buc, 455
rescode.err_nan_in_bux, 456
rescode.err_nan_in_c, 455
rescode.err_nan_in_double_data, 455
rescode.err_negative_append, 456
rescode.err_negative_surplus, 456
rescode.err_newer_dll, 446
rescode.err_no_bars_for_solution, 458
rescode.err_no_barx_for_solution, 458
rescode.err_no_basis_sol, 456
rescode.err_no_doty, 464
rescode.err_no_dual_for_itg_sol, 457
rescode.err_no_dual_infeas_cer, 457
rescode.err_no_init_env, 447
rescode.err_no_optimizer_var_type, 456
rescode.err_no_primal_infeas_cer, 457
rescode.err_no_snx_for_bas_sol, 457
rescode.err_no_solution_in_callback, 457
rescode.err_non_unique_array, 459
rescode.err_nonconvex, 453
rescode.err_nonlinear_equality, 453
rescode.err_nonlinear_ranged, 453
rescode.err_not_power_domain, 463
rescode.err_null_env, 447
rescode.err_null_pointer, 447
rescode.err_null_task, 447
rescode.err_num_arguments, 451
rescode.err_numconlim, 452
rescode.err_numvarlim, 452
rescode.err_obj_q_not_nsd, 454
rescode.err_obj_q_not_psd, 454
rescode.err_objective_range, 453
rescode.err_older_dll, 446
rescode.err_opf_dual_integer_solution, 449
rescode.err_opf_duplicate_bound, 449
rescode.err_opf_duplicate_cone_entry, 449
rescode.err_opf_duplicate_constraint_name, 449
rescode.err_opf_incorrect_tag_param, 449
rescode.err_opf_invalid_cone_type, 449
rescode.err_opf_invalid_tag, 449
rescode.err_opf_mismatched_tag, 449
rescode.err_opf_premature_eof, 449
rescode.err_opf_syntax, 449
rescode.err_opf_too_large, 449
rescode.err_optimizer_license, 445
rescode.err_overflow, 456
rescode.err_param_index, 451
rescode.err_param_is_too_large, 451
rescode.err_param_is_too_small, 451
rescode.err_param_name, 451
rescode.err_param_name_dou, 451
rescode.err_param_name_int, 451
rescode.err_param_name_str, 451
rescode.err_param_type, 451
rescode.err_param_value_str, 451
rescode.err_platform_not_licensed, 445
rescode.err_postsolve, 456
rescode.err_pro_item, 453
rescode.err_prob_license, 445
rescode.err_ptf_format, 450
rescode.err_ptf_incompatibility, 450
rescode.err_ptf_inconsistency, 450
rescode.err_ptf_undefined_item, 450
rescode.err_qcon_subi_too_large, 455
rescode.err_qcon_subi_too_small, 455
rescode.err_qcon_upper_triangle, 455
rescode.err_qobj_upper_triangle, 455
rescode.err_read_async, 447
rescode.err_read_format, 448
rescode.err_read_gzip, 447
rescode.err_read_lp_delayed_rows_not_supported, 450
rescode.err_read_lp_missing_end_tag, 450
rescode.err_read_premature_eof, 448

rescode.err_read_write, 457
rescode.err_read_zstd, 447
rescode.err_remove_cone_variable, 454
rescode.err_repair_invalid_problem, 457
rescode.err_repair_optimization_failed, 457
rescode.err_sen_bound_invalid_lo, 458
rescode.err_sen_bound_invalid_up, 458
rescode.err_sen_format, 458
rescode.err_sen_index_invalid, 458
rescode.err_sen_index_range, 458
rescode.err_sen_invalid_regexp, 458
rescode.err_sen_numerical, 458
rescode.err_sen_solution_status, 458
rescode.err_sen_undef_name, 458
rescode.err_sen_unhandled_problem_type, 458
rescode.err_server_access_token, 463
rescode.err_server_address, 463
rescode.err_server_certificate, 463
rescode.err_server_connect, 463
rescode.err_server_hard_timeout, 463
rescode.err_server_problem_size, 463
rescode.err_server_protocol, 463
rescode.err_server_status, 463
rescode.err_server_tls_client, 463
rescode.err_server_token, 463
rescode.err_shape_is_too_large, 451
rescode.err_size_license, 445
rescode.err_size_license_con, 445
rescode.err_size_license_intvar, 445
rescode.err_size_license_var, 445
rescode.err_slice_size, 456
rescode.err_sol_file_invalid_number, 454
rescode.err_solitem, 452
rescode.err_solver_probtype, 453
rescode.err_space, 446
rescode.err_space_leaking, 447
rescode.err_space_no_info, 447
rescode.err_sparsity_specification, 447
rescode.err_sym_mat_duplicate, 459
rescode.err_sym_mat_huge, 456
rescode.err_sym_mat_invalid, 456
rescode.err_sym_mat_invalid_col_index, 459
rescode.err_sym_mat_invalid_row_index, 459
rescode.err_sym_mat_invalid_value, 459
rescode.err_sym_mat_not_lower_tringular, 459
rescode.err_task_incompatible, 457
rescode.err_task_invalid, 457
rescode.err_task_premature_eof, 457
rescode.err_task_write, 457
rescode.err_thread_cond_init, 446
rescode.err_thread_create, 446
rescode.err_thread_mutex_init, 446
rescode.err_thread_mutex_lock, 446
rescode.err_thread_mutex_unlock, 446
rescode.err_toconic_constr_not_conic, 462
rescode.err_toconic_constr_q_not_psd, 462
rescode.err_toconic_constraint_fx, 462
rescode.err_toconic_constraint_ra, 462
rescode.err_toconic_objective_not_psd, 462
rescode.err_too_small_a_truncation_value, 455
rescode.err_too_small_max_num_nz, 452
rescode.err_too_small_maxnumanz, 452
rescode.err_unallowed_whichsol, 452
rescode.err_unb_step_size, 458
rescode.err_undef_solution, 464
rescode.err_undefined_objective_sense, 455
rescode.err_unhandled_solution_status, 460
rescode.err_unknown, 446
rescode.err_upper_bound_is_a_nan, 455
rescode.err_upper_triangle, 460
rescode.err_whichitem_not_allowed, 452
rescode.err_whichsol, 452
rescode.err_write_async, 450
rescode.err_write_lp_duplicate_con_names, 448
rescode.err_write_lp_duplicate_var_names, 448
rescode.err_write_lp_invalid_con_names, 448
rescode.err_write_lp_invalid_var_names, 448
rescode.err_write_mps_invalid_name, 449
rescode.err_write_opf_invalid_var_name, 450
rescode.err_writing_file, 450
rescode.err_y_is_undefined, 455

Index

A

ACC, 21
affine conic constraints, 21
analysis
    infeasibility, 194
asset, *see* portfolio optimization
attaching
    streams, 19

B

basic
    solution, 81
basis identification, 110, 172
basis type
    sensitivity analysis, 200
big-M, 190
BLAS, 117
bound
    constraint, 15, 156, 159, 163
    linear optimization, 15
    variable, 15, 156, 159, 163
Branch-and-Bound, 181

C

callback, 90
cardinality constraints, 146
CBF format, 527
ceo1
    example, 37
certificate, 82
    dual, 158, 161
    infeasibility, 77
    infeasible, 77
    primal, 158, 161
Cholesky factorization, 119, 135
column ordered
    matrix format, 209
complementarity, 157, 161
concurrent optimizer, 152
cone
    dual, 160
    dual exponential, 36
    exponential, 36
    power, 33
    quadratic, 29
    rotated quadratic, 29
    semidefinite, 43
conic exponential optimization, 36
conic optimization, 21, 29, 33, 36, 159
    interior-point, 176
    mixed-integer, 189
    termination criteria, 178
conic problem
    example, 29, 33, 37
conic quadratic optimization, 29
constraint
    bound, 15, 156, 159, 163
    linear optimization, 15
    matrix, 15, 156, 159, 163
    quadratic, 164
constraint programming, 58
correlation matrix, 128
covariance matrix, *see* correlation matrix
cqo1
    example, 29
cuts, 187
cutting planes, 187

D

defining
    objective, 19
determinism, 123
disjunction, 58
disjunctive constraint, 190
disjunctive constraints, 58
DJC, 58
domain, 497
dual
    certificate, 158, 161
    cone, 160
    feasible, 157
    infeasible, 157, 158, 161
    problem, 157, 160, 164
    solution, 83
    variable, 157, 160
duality
    conic, 160
    linear, 157
    semidefinite, 164
dualizer, 168

E

efficient frontier, 132
eliminator, 168
error
    optimization, 81
errors, 84
example

ceo1, 37
conic problem, 29, 33, 37
cqo1, 29
lo1, 19
pow1, 33
qo1, 65
quadratic objective, 65
exceptions, 84
exponential cone, 36

F

factor model, 135
feasibility
integer feasibility, 183
feasible
dual, 157
primal, 156, 170, 177
problem, 156
format, 87
CBF, 527
json, 553
LP, 501
MPS, 505
OPF, 517
PTF, 545
sol, 559
task, 553
full
vector format, 208

G

geometric programming, 40
GP, 40

H

heuristic, 186
hot-start, 174

I

I/O, 87
infeasibility, 82, 158, 161
analysis, 194
linear optimization, 158
repair, 194
semidefinite, 164
infeasibility certificate, 77
infeasible
dual, 157, 158, 161
primal, 156, 158, 161, 170, 177
problem, 156, 158, 164
information item, 89, 90
installation, 9
PIP, 10
requirements, 9
setup script, 10
troubleshooting, 9
integer
solution, 81
variable, 53
integer feasibility, 183
feasibility, 183
integer optimization, 53, 58
initial solution, 56
parameter, 53
interior-point
conic optimization, 176
linear optimization, 170
logging, 173, 179
optimizer, 170, 176
solution, 81
termination criteria, 171, 178

J

json format, 553

L

LAPACK, 117
license, 125
linear
objective, 19
linear constraint matrix, 15
linear dependency, 168
linear optimization, 15, 156
bound, 15
constraint, 15
infeasibility, 158
interior-point, 170
objective, 15
simplex, 174
termination criteria, 171, 174
variable, 15
linearity interval, 199
lo1
example, 19
log-sum-exp, 149
logging, 86
interior-point, 173, 179
mixed-integer optimizer, 184
optimizer, 173, 175, 179
simplex, 175
logistic regression, 148
LP format, 501

M

machine learning
logistic regression, 148
market impact cost, 140
Markowitz model, 128
matrix
constraint, 15, 156, 159, 163
semidefinite, 43
symmetric, 43
matrix format
column ordered, 209
row ordered, 209
triplets, 209

memory management, 122
MI(QC)QO, 190
MICO, 189
MIP, *see* integer optimization
mixed-integer, *see* integer
    conic optimization, 189
    optimizer, 180
    presolve, 186
    quadratic, 190
mixed-integer optimization, *see* integer optimization, 180
mixed-integer optimizer
    logging, 184
modeling
    design, 11
MPS format, 505
    free, 517

N

numerical issues
    presolve, 168
    scaling, 169
    simplex, 175
numpy, 123

O

objective, 156, 159, 163
    defining, 19
    linear, 19
    linear optimization, 15
OPF format, 517
optimal
    solution, 82
optimality gap, 182
optimization
    conic, 21, 159
    conic quadratic, 159
    error, 81
    integer, 58
    linear, 15, 156
    semidefinite, 162
optimizer
    concurrent, 152
    conic, 21
    determinism, 123
    interior-point, 170, 176
    interrupt, 90
    logging, 173, 175, 179
    mixed-integer, 58, 180
    parallel, 76
    selection, 168, 169
    simplex, 174
    termination, 182

P

parallel optimization, 76, 152
parallelization, 123
parameter, 88
    integer optimization, 53
    simplex, 175
Pareto optimality, 128
portfolio optimization, 127
    cardinality constraints, 146
    efficient frontier, 132
    factor model, 135
    market impact cost, 140
    Markowitz model, 128
    Pareto optimality, 128
    slippage cost, 139
    transaction cost, 143
positive semidefinite, 65
pow1
    example, 33
power cone, 33
power cone optimization, 33
presolve, 167
    eliminator, 168
    linear dependency check, 168
    mixed-integer, 186
    numerical issues, 168
primal
    certificate, 158, 161
    feasible, 156, 170, 177
    infeasible, 156, 158, 161, 170, 177
    problem, 157, 160, 164
    solution, 83, 156
primal heuristics, 186
primal-dual
    problem, 170, 176
    solution, 157
problem
    dual, 157, 160, 164
    feasible, 156
    infeasible, 156, 158, 164
    load, 87
    primal, 157, 160, 164
    primal-dual, 170, 176
    save, 87
    status, 81
    unbounded, 158, 162
PTF format, 545

Q

qo1
    example, 65
quadratic
    constraint, 164
    mixed-integer, 190
quadratic cone, 29
quadratic objective
    example, 65
quadratic optimization, 164

R

regression
    logistic, 148

relaxation, 181
repair
    infeasibility, 194
response code, 84
rotated quadratic cone, 29
row ordered
    matrix format, 209

S

scaling, 169
semidefinite
    cone, 43
    infeasibility, 164
    matrix, 43
    variable, 43
semidefinite optimization, 43, 162
sensitivity analysis, 198
    basis type, 200
setup script, 10
shadow price, 199
simplex
    linear optimization, 174
    logging, 175
    numerical issues, 175
    optimizer, 174
    parameter, 175
    termination criteria, 174
slippage cost, 139
sol format, 559
solution
    basic, 81
    dual, 83
    file format, 559
    integer, 81
    interior-point, 81
    optimal, 82
    primal, 83, 156
    primal-dual, 157
    retrieve, 81
    status, 18, 82
solving linear system, 114
sparse
    vector format, 208
sparse vector, 208
status
    problem, 81
    solution, 18, 82
streams
    attaching, 19
symmetric
    matrix, 43

T

task format, 553
termination, 81
    optimizer, 182
termination criteria, 90, 182
    conic optimization, 178
    interior-point, 171, 178
    linear optimization, 171, 174
    simplex, 174
    tolerance, 172, 179
thread, 123
time, 123
time limit, 90
timing, 123
tolerance
    termination criteria, 172, 179
transaction cost, 143
triplets
    matrix format, 209
troubleshooting
    installation, 9

U

unbounded
    problem, 158, 162
user callback, *see* callback

V

valid inequalities, 187
variable, 156, 159, 163
    bound, 15, 156, 159, 163
    dual, 157, 160
    integer, 53
    linear optimization, 15
    semidefinite, 43
vector format
    full, 208
    sparse, 208