



MOSEK Optimizer API for .NET

*Release 9.3.21*

MOSEK ApS

08 August 2022

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Why the Optimizer API for .NET? . . . . .	2
<b>2</b>	<b>Contact Information</b>	<b>3</b>
<b>3</b>	<b>License Agreement</b>	<b>4</b>
<b>4</b>	<b>Installation</b>	<b>8</b>
4.1	.NET Core . . . . .	8
4.2	Manual installation . . . . .	8
4.3	Testing the Installation and Compiling Examples . . . . .	9
4.4	Other platforms: .NET Core, Mono . . . . .	9
<b>5</b>	<b>Design Overview</b>	<b>11</b>
5.1	Modeling . . . . .	11
5.2	“Hello World!” in <b>MOSEK</b> . . . . .	11
<b>6</b>	<b>Optimization Tutorials</b>	<b>13</b>
6.1	Linear Optimization . . . . .	13
6.2	Quadratic Optimization . . . . .	20
6.3	Conic Quadratic Optimization . . . . .	28
6.4	Power Cone Optimization . . . . .	33
6.5	Conic Exponential Optimization . . . . .	36
6.6	Semidefinite Optimization . . . . .	40
6.7	Integer Optimization . . . . .	47
6.8	Geometric Programming . . . . .	53
6.9	Library of basic functions . . . . .	56
6.10	Problem Modification and Reoptimization . . . . .	65
6.11	Parallel optimization . . . . .	70
<b>7</b>	<b>Solver Interaction Tutorials</b>	<b>73</b>
7.1	Accessing the solution . . . . .	73
7.2	Errors and exceptions . . . . .	77
7.3	Input/Output . . . . .	79
7.4	Setting solver parameters . . . . .	81
7.5	Retrieving information items . . . . .	83
7.6	Progress and data callback . . . . .	83
7.7	<b>MOSEK</b> OptServer . . . . .	87
<b>8</b>	<b>Debugging Tutorials</b>	<b>91</b>
8.1	Understanding optimizer log . . . . .	92
8.2	Addressing numerical issues . . . . .	96
8.3	Debugging infeasibility . . . . .	98
8.4	Python Console . . . . .	103
<b>9</b>	<b>Advanced Numerical Tutorials</b>	<b>105</b>
9.1	Solving Linear Systems Involving the Basis Matrix . . . . .	105
9.2	Calling BLAS/LAPACK Routines from <b>MOSEK</b> . . . . .	114



18.2	Functions . . . . .	483
18.3	Parameters . . . . .	485
18.4	Constants . . . . .	486
18.5	Response Codes . . . . .	487
<b>Bibliography</b>		<b>490</b>
<b>Symbol Index</b>		<b>491</b>
<b>Index</b>		<b>506</b>

# Chapter 1

## Introduction

The **MOSEK** Optimization Suite 9.3.21 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

$$Ax - b \geq 0.$$

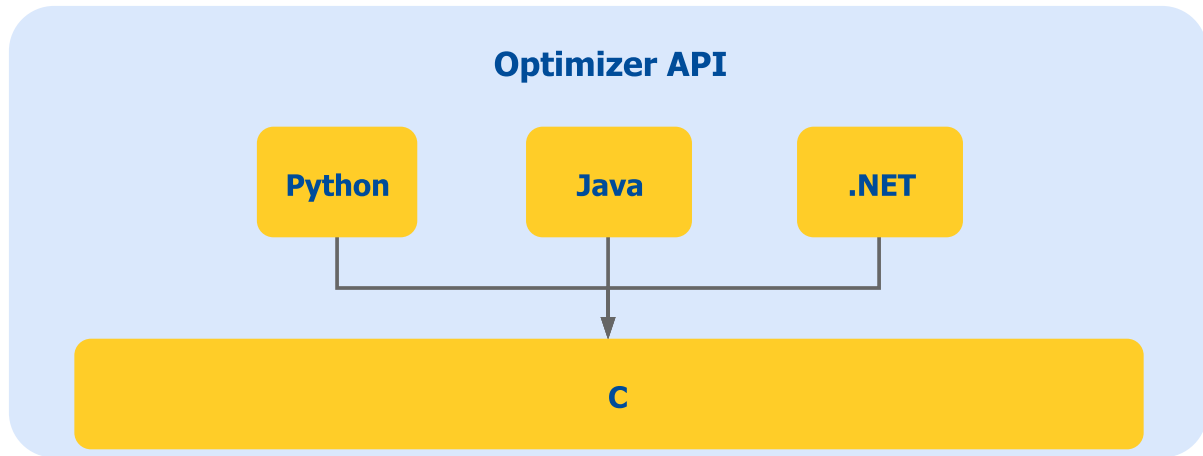
In conic optimization this is replaced with a wider class of constraints

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

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 .NET?

The Optimizer API for .NET 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 .NET can be used with any application running on the Microsoft .NET platform (and other .NET implementations like Mono and .NET Core). It consists of a single library, `mosekdotnet9_3.dll`, containing classes and more in the `mosek` namespace.

The Optimizer API for .NET 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)

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	
Website	<a href="http://mosek.com">mosek.com</a>	
Email		
	<a href="mailto:sales@mosek.com">sales@mosek.com</a>	Sales, pricing, and licensing
	<a href="mailto:support@mosek.com">support@mosek.com</a>	Technical support, questions and bug reports
	<a href="mailto:info@mosek.com">info@mosek.com</a>	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:

<b>Blogger</b>	<a href="https://blog.mosek.com/">https://blog.mosek.com/</a>
<b>Google Group</b>	<a href="https://groups.google.com/forum/#!forum/mosek">https://groups.google.com/forum/#!forum/mosek</a>
<b>Twitter</b>	<a href="https://twitter.com/mosektw">https://twitter.com/mosektw</a>
<b>Linkedin</b>	<a href="https://www.linkedin.com/company/mosek-aps">https://www.linkedin.com/company/mosek-aps</a>
<b>Youtube</b>	<a href="https://www.youtube.com/channel/UCvIyectEVLp31NXeD5mIbEw">https://www.youtube.com/channel/UCvIyectEVLp31NXeD5mIbEw</a>

In particular **Twitter** is used for news, updates and release announcements.

## Chapter 3

# License Agreement

Before using the **MOSEK** software, please read the license agreement available in the distribution at <MSKHOME>/mosek/9.3/mosek-eula.pdf or on the **MOSEK** website <https://mosek.com/products/license-agreement>.

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

### ***zlib***

**MOSEK** includes 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 1.2.7, May 2nd, 2012

Copyright (C) 1995-2012 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** includes 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.
 *
 *****/
```

## *Zstandard*

**MOSEK** includes the *Zstandard* library developed by Facebook obtained from [github/zstd](#). The license agreement for *Zstandard* is shown in [Listing 3.3](#).

Listing 3.3: *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
```

(continues on next page)

(continued from previous page)

(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.

### *mimalloc*

**MOSEK** includes the *mimalloc* memory allocator library from [github/mimalloc](https://github.com/mimalloc). The license agreement for *mimalloc* is shown in [Listing 3.4](#).

Listing 3.4: *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.

### *blis*

**MOSEK** includes the *blis* implementation of BLAS library from [github/blis](https://github.com/blis). The license agreement for *blis* is shown in [Listing 3.5](#).

Listing 3.5: *blis* license.

Copyright (C) 2018, The University of Texas at Austin  
Copyright (C) 2016, Hewlett Packard Enterprise Development LP  
Copyright (C) 2018 - 2019, Advanced Micro Devices, Inc.

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(s) of the copyright holder(s) nor the names of its contributors may be used to endorse or promote products derived

(continues on next page)

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.

### **OpenBLAS**

**MOSEK** includes the *OpenBLAS* implementation of BLAS library from [github/OpenBLAS](https://github.com/OpenBLAS). The license agreement for *OpenBLAS* is shown in [Listing 3.6](#).

Listing 3.6: *openblas* license.

Copyright (c) 2011-2014, The OpenBLAS 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 the documentation and/or other materials provided with the distribution.
3. Neither the name of the OpenBLAS project 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 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.

# Chapter 4

## Installation

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

---

**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 .NET is compatible with the Microsoft .NET framework version 4.5 and later, Mono v.1.2 and later, and .NET Standard 2.0 and later.

## 4.1 .NET Core

The Optimizer API for .NET can be installed as a cross-platform .NET Core package. The NuGet package `Mosek.9.3.21.nupkg` is available for download from:

- our website <https://mosek.com/downloads>
- the NuGet repository <https://www.nuget.org/packages/Mosek/>

Follow the instructions for your .NET Core toolchain to install the package from the repository. The package is available on all platforms.

## 4.2 Manual installation

### Locating files in the MOSEK Optimization Suite

The relevant files of the Optimizer API for .NET are organized as reported in [Table 4.1](#).

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

Relative Path	Description	Label
<MSKHOME>/mosek/9.3/tools/platform/<PLATFORM>/bin	Libraries	<LIBDIR>
<MSKHOME>/mosek/9.3/tools/examples/dotnet	Examples	<EXDIR>
<MSKHOME>/mosek/9.3/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 the **MOSEK**, i.e. `win32x86`, `win64x86`, `linux64x86` and `linuxaarch64`.

## Setting up paths

To compile a .NET program using **MOSEK** the correct path to `mosekdotnet.dll` must be provided. For example, using the Microsoft .NET compiler this is done with the command line option

```
csc /r:"<LIBDIR>\mosekdotnet.dll" lo1.cs
```

To run applications the system must be able to locate `mosekdotnet.dll`, either in the current directory or in the Global Assembly Cache.

## 4.3 Testing the Installation and Compiling Examples

This section describes how to verify that **MOSEK** has been installed correctly, and how to build and execute the .NET examples distributed with **MOSEK**.

### Compiling and running from the command line

To compile an example, say `lo1`, with the Microsoft .NET compiler, open a DOS box with paths for Visual Studio set up (usually in the Start menu, the sub-menu for Visual Studio contains an entry that starts a DOS box with everything set up).

To compile the example `lo1.cs` distributed with **MOSEK**:

- Go to the examples directory `<EXDIR>`.
- To compile the code and produce an executable, type:

```
csc /r:"<LIBDIR>\mosekdotnet.dll" lo1.cs
```

or for Visual Basic:

```
vbc /r:"<LIBDIR>\mosekdotnet.dll" lo1.vb
```

- Copy `mosekdotnet.dll` into the directory where `lo1.exe` was created, and run the program with:

```
lo1
```

### Compiling the examples using `nmake`

A makefile for use with `nmake`, named `Makefile` is available in `<EXDIR>`. To compile all examples using this makefile use the command

```
make /f Makefile all
```

## 4.4 Other platforms: .NET Core, Mono

The library `mosekdotnet.dll` may be used from any .NET compatible language such as Visual Basic, Microsoft C# or Microsoft Managed C++ and with .NET Core and Mono. Both the examples and the library should also work with Mono on most 32-bit platforms. If the file `mosekdotnet.dll` is not included in the **MOSEK** distribution for your platform, use `mosekdotnet.dll` included in the Windows distribution.

Note that the library accesses methods in the native **MOSEK** library, which is considered *unsafe* from a .NET point of view. This means that use of the library in certain restricted contexts is not possible — building an ordinary application and running it from a local drive should not be a problem.

#### 4.4.1 Mono

Mono is a free implementation of the .NET platform available at <http://mono-project.com/>. To use it install **MOSEK** as described in the [Installation Manual](#). Set the environment variable

```
MONO_PATH
```

to point to `mosekdotnet9_3.dll` for the 64-bit Mono. You should now be able to compile and run the distributed .NET examples using Mono.

#### 4.4.2 IronPython

It is possible to use the **MOSEK** .NET API interactively from .NET languages which implement a command-line interpreter, for example IronPython, available at <http://ironpython.net/>. This can be used to create and examine the problems and solutions from **MOSEK** more easily.

#### 4.4.3 MOSEK and .NET Core

The **MOSEK** NuGet package `Mosek.9.3.21.nupkg` is a complete cross-platform .NET Core compatible distribution that works on Windows, Linux and OS X. Assuming that the `Mosek.9.3.21.nupkg` file has been downloaded in a directory `local-nupkgs`, modify the configuration file `*.csproj` to add the following entry

```
<PropertyGroup>
  <RestoreSources>$(RestoreSources);local-nupkgs</RestoreSources>
</PropertyGroup>
```

Now, add the dependency on **MOSEK** to the project:

```
dotnet add package Mosek
```

and the project using **MOSEK** API can be built:

```
dotnet build
dotnet run
```

Installation instructions for different .NET Core compatible environments may vary.

# Chapter 5

## Design Overview

### 5.1 Modeling

Optimizer API for .NET 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 .NET 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 .NET.

#### Creating an environment and task

Every interaction with **MOSEK** using Optimizer API for .NET begins 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. An environment can host multiple tasks.

## 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

```
using mosek;
using System;

public class helloworld {
    public static void Main() {

        double[] x = new double[1];

        using (Env env = new Env()) {           // Create Environment
            using (Task task = new Task(env, 0, 1)) { // 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

                task.getxx(soltype.itr, x);        // Get solution
                Console.WriteLine("Solution x = " + 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.

### 6.1 Linear Optimization

The simplest optimization problem is a purely linear problem. A *linear optimization problem* 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

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

and the bounds

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

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

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

- $l^c$  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:

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

under the bounds

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

### Solving the problem

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

1. Create an environment.
2. Create an optimization task.
3. Load a problem into the task object.
4. Optimization.
5. Extracting the solution.

Below we explain each of these steps.

#### Create an environment.

Before setting up the optimization problem, a **MOSEK** environment must be created. All tasks in the program should share the same environment.

```
// Make mosek environment.
using (mosek.Env env = new mosek.Env())
{
```

#### Create an optimization task.

Next, an empty task object is created:

```
// Create a task object.
using (mosek.Task task = new mosek.Task(env, 0, 0))
{
    // Directs the log task stream to the user specified
    // method msgclass.streamCB
    task.set_Stream (mosek.streamtype.log, new msgclass (""));
```

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. 7.3](#).



and

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

### 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 in the arrays:

```
int[] [] asub = {
    new int[] {0, 1},
    new int[] {0, 1, 2},
    new int[] {0, 1},
    new int[] {1, 2}
};
double[] [] aval = {
    new double[] {3.0, 2.0},
    new double[] {1.0, 1.0, 2.0},
    new double[] {2.0, 3.0},
    new double[] {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.

Using the function `Task.putacol` we set column  $j$  of  $A$

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

There are many alternative formats for entering the  $A$  matrix. See functions such as `Task.putarow`, `Task.putarowlist`, `Task.putaijlist` and similar.

Finally, the bounds on each constraint are set by looping over each constraint index  $i = 0, \dots, \text{numcon} - 1$

```
// Set the bounds on constraints.
// blc[i] <= constraint_i <= buc[i]
for (int i = 0; i < numcon; ++i)
    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`. If the solution status is reported as `solsta.optimal` the solution is extracted in the lines below:

```
task.getxx(mosek.soltype.bas, // Request the basic solution.
          xx);
```

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`.

## Catching exceptions

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

```
catch (mosek.Exception e) {
    mosek.rescode res = e.Code;
    Console.WriteLine("Response code {0}\nMessage      {1}", res, e.Message);
}
```

The types of exceptions that **MOSEK** can throw can be seen in [Sec. 15.5](#). See also [Sec. 7.2](#).

## Source code

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

Listing 6.1: Linear optimization example.

```
using System;

namespace mosek.example
{
    class msgclass : mosek.Stream
    {
        string prefix;
        public msgclass (string prfx)
        {
            prefix = prfx;
        }

        public override void streamCB (string msg)
        {
            Console.Write ("{0}{1}", prefix, msg);
        }
    }

    public class lo1
    {
        public static void Main ()
        {
            const int numcon = 3;
            const int numvar = 4;

            // Since the value of infinity is ignored, we define it solely
            // for symbolic purposes
            double infinity = 0;

            double[] c    = {3.0, 1.0, 5.0, 1.0};
```

(continues on next page)

```

int[] [] asub = {
    new int[] {0, 1},
    new int[] {0, 1, 2},
    new int[] {0, 1},
    new int[] {1, 2}
};
double[] [] aval = {
    new double[] {3.0, 2.0},
    new double[] {1.0, 1.0, 2.0},
    new double[] {2.0, 3.0},
    new double[] {1.0, 3.0}
};

mosek.bindkey[] bkc = {mosek.bindkey.fx,
                      mosek.bindkey.lo,
                      mosek.bindkey.up
                      };

double[] blc = {30.0,
                15.0,
                -infinity
                };
double[] buc = {30.0,
                +infinity,
                25.0
                };
mosek.bindkey[] bkc = {mosek.bindkey.lo,
                      mosek.bindkey.ra,
                      mosek.bindkey.lo,
                      mosek.bindkey.lo
                      };

double[] blx = {0.0,
                0.0,
                0.0,
                0.0
                };
double[] bux = {+infinity,
                10.0,
                +infinity,
                +infinity
                };

try
{
    // Make mosek environment.
    using (mosek.Env env = new mosek.Env())
    {
        // Create a task object.
        using (mosek.Task task = new mosek.Task(env, 0, 0))
        {
            // Directs the log task stream to the user specified
            // method msgclass.streamCB
            task.set_Stream (mosek.streamtype.log, new msgclass (""));

            // Append 'numcon' empty constraints.
            // The constraints will initially have no bounds.

```

(continues on next page)

```

task.appendcons(numcon);

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

for (int j = 0; j < numvar; ++j)
{
    // 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, bxx[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 (int i = 0; i < numcon; ++i)
    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
mosek.solsta solsta;

task.getsolsta(mosek.soltype.bas, out solsta);

switch (solsta)
{
    case mosek.solsta.optimal:
        double[] xx = new double[numvar];
        task.getxx(mosek.soltype.bas, // Request the basic solution.
                  xx);

        Console.WriteLine ("Optimal primal solution\n");
        for (int j = 0; j < numvar; ++j)
            Console.WriteLine ("x[{0}]: {1}", j, xx[j]);
        break;
    case mosek.solsta.dual_infeas_cer:
    case mosek.solsta.prim_infeas_cer:
        Console.WriteLine("Primal or dual infeasibility certificate found.\n
→");

```

(continues on next page)





```

class msgclass : mosek.Stream
{
    string prefix;
    public msgclass (string prfx)
    {
        prefix = prfx;
    }

    public override void streamCB (string msg)
    {
        Console.Write ("{0}{1}", prefix, msg);
    }
}

public class qo1
{
    public static void Main ()
    {
        // Since the value infinity is never used, we define
        // 'infinity' symbolic purposes only
        const double infinity = 0;
        const int numcon = 1;    /* Number of constraints.          */
        const int numvar = 3;    /* Number of variables.      */

        double[] c = {0.0, -1.0, 0.0};

        mosek.boundkey[] bkc = {mosek.boundkey.lo};
        double[] blc = {1.0};
        double[] buc = {infinity};

        mosek.boundkey[] bkc = {mosek.boundkey.lo,
                                mosek.boundkey.lo,
                                mosek.boundkey.lo
                                };

        double[] blx = {0.0,
                        0.0,
                        0.0
                        };
        double[] bux = {+infinity,
                        +infinity,
                        +infinity
                        };

        int[][] asub = { new int[] {0}, new int[] {0}, new int[] {0}};
        double[][] aval = { new double[] {1.0}, new double[] {1.0}, new double[] {1.0}}
→;

        mosek.Task task = null;
        mosek.Env env = null;
        double[] xx = new double[numvar];
        try
        {
            // Make mosek environment.
            env = new mosek.Env ();
            // Create a task object linked with the environment env.
            task = new mosek.Task (env, 0, 0);

```

(continues on next page)

```

// Directs the log task stream to the user specified
// method task_msg_obj.streamCB
task.set_Stream (mosek.streamtype.log, new msgclass (""));

/* Give MOSEK an estimate of the size of the input data.
   This is done to increase the speed of inputting data.
   However, it is optional. */
/* 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 (int j = 0; j < numvar; ++j)
{
    /* 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, bkc[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.
   for i=1, ..., numcon : blc[i] <= constraint i <= buc[i] */
for (int i = 0; i < numcon; ++i)
    task.putconbound(i, bkc[i], blc[i], buc[i]);

/*
 * The lower triangular part of the Q
 * matrix in the objective is specified.
 */

int[]    qsubi = {0, 1, 2, 2 };
int[]    qsubj = {0, 1, 0, 2 };
double[] qval = {2.0, 0.2, -1.0, 2.0};

/* Input the Q for the objective. */

task.putobjsense(mosek.objsense.minimize);

task.putqobj(qsubi, qsubj, qval);

task.optimize();

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

mosek.solsta solsta;
/* Get status information about the solution */

```

```

task.getsolsta(mosek.soltype.itr, out solsta);
switch (solsta)
{
    case mosek.solsta.optimal:
        task.getxx(mosek.soltype.itr, // Interior point solution.
            xx);

        Console.WriteLine ("Optimal primal solution\n");
        for (int j = 0; j < numvar; ++j)
            Console.WriteLine ("x[{0}]:", xx[j]);
        break;
    case mosek.solsta.dual_infeas_cer:
    case mosek.solsta.prim_infeas_cer:
        Console.WriteLine("Primal or dual infeasibility.\n");
        break;
    case mosek.solsta.unknown:
        Console.WriteLine("Unknown solution status.\n");
        break;
    default:
        Console.WriteLine("Other solution status");
        break;
}
}
catch (mosek.Exception e)
{
    Console.WriteLine (e);
    throw;
}
finally
{
    if (task != null) task.Dispose ();
    if (env != null) env.Dispose ();
}
} /* Main */
}

```

### 6.2.2 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 (6.3).

Consider the problem:

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

This is equivalent to

$$\begin{aligned}
 & \text{minimize} && \frac{1}{2}x^T Q^o x + c^T x \\
 & \text{subject to} && \frac{1}{2}x^T Q^0 x + Ax \geq b, \\
 & && x \geq 0,
 \end{aligned} \tag{6.5}$$

where

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

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

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`.

```
int[]
qsubi = { 0, 1, 2, 2 },
qsubj = { 0, 1, 2, 0 };
double[]
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 6.3: Implementation of the quadratically constrained problem (6.5).

```
using System;

namespace mosek.example
{
    class msgclass : mosek.Stream
    {
        string prefix;
        public msgclass (string prfx)
        {
            prefix = prfx;
        }

        public override void streamCB (string msg)
        {
            Console.Write ("{0}{1}", prefix, msg);
        }
    }

    public class qcqo1
    {
        public static void Main ()
        {
            const double inf = 0.0; /* We don't actually need any value for infinity */

            const int numcon = 1;    /* Number of constraints. */
            const int numvar = 3;    /* Number of variables. */

            mosek.boundkey[]
```

(continues on next page)

```

bkc = { mosek.boundkey.lo },
bkc = { mosek.boundkey.lo, mosek.boundkey.lo, mosek.boundkey.lo };
int[] [] asub = { new int[] {0}, new int[] {0}, new int[] {0} };
double[] [] aval = { new double[] {1.0}, new double[] {1.0}, new double[] {1.0} };

double[]
blc = { 1.0 },
buc = { inf },
c = { 0.0, -1.0, 0.0 },
blx = { 0.0, 0.0, 0.0 },
bux = { inf, inf, inf },
xx = new double[numvar];
try
{
    using (mosek.Env env = new mosek.Env())
    {
        using (mosek.Task task = new mosek.Task(env))
        {
            task.set_Stream (mosek.streamtype.log, new msgclass (""));

            /* Give MOSEK an estimate of the size of the input data.
               This is done to increase the speed of inputting data.
               However, it is optional. */

            /* 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 (int j = 0; j < numvar; ++j)
            {
                /* 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, bkc[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 */
                           aval[j]);                             /* Non-zero Values of column j. */
            }
            /* Set the bounds on constraints.
               for i=1, ..., numcon : blc[i] <= constraint i <= buc[i] */
            for (int i = 0; i < numcon; ++i)
                task.putconbound(i, bkc[i], blc[i], buc[i]);
            /*
             * The lower triangular part of the Q
             * matrix in the objective is specified.
             */

            {
                int[]

```

```

qsubi = { 0, 1, 2, 2 },
qsubj = { 0, 1, 0, 2 };
double[]
qval = { 2.0, 0.2, -1.0, 2.0 };

/* Input the Q for the objective. */

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
*/
{
    int[]
    qsubi = { 0, 1, 2, 2 },
    qsubj = { 0, 1, 2, 0 };
    double[]
    qval = { -2.0, -2.0, -0.2, 0.2 };

    /* put Q^0 in constraint with index 0. */

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

task.putobjsense(mosek.objsense.minimize);

task.optimize();

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

mosek.solsta solsta;
/* Get status information about the solution */
task.getsolsta(mosek.soltype.itr, out solsta);

task.getxx(mosek.soltype.itr, // Basic solution.
           xx);

switch (solsta)
{
    case mosek.solsta.optimal:
        Console.WriteLine ("Optimal primal solution\n");
        for (int j = 0; j < numvar; ++j)
            Console.WriteLine ("x[{0}]:", xx[j]);
        break;
    case mosek.solsta.dual_infeas_cer:
    case mosek.solsta.prim_infeas_cer:
        Console.WriteLine("Primal or dual infeasibility.\n");
        break;
}

```





```

    prefix = prfx;
}

public override void streamCB (string msg)
{
    Console.Write ("{0}{1}", prefix, msg);
}
}

public class cq01
{
    public static void Main ()
    {
        const int numcon = 1;
        const int numvar = 6;

        // Since the value infinity is never used, we define
        // 'infinity' symbolic purposes only
        double infinity = 0;

        mosek.boundkey[] bkc    = { mosek.boundkey.fx };
        double[] blc = { 1.0 };
        double[] buc = { 1.0 };

        mosek.boundkey[] bkc = {mosek.boundkey.lo,
                                mosek.boundkey.lo,
                                mosek.boundkey.lo,
                                mosek.boundkey.fr,
                                mosek.boundkey.fr,
                                mosek.boundkey.fr
                                };

        double[] blx = { 0.0,
                        0.0,
                        0.0,
                        -infinity,
                        -infinity,
                        -infinity
                        };
        double[] bux = { +infinity,
                        +infinity,
                        +infinity,
                        +infinity,
                        +infinity,
                        +infinity
                        };

        double[] c    = { 0.0,
                        0.0,
                        0.0,
                        1.0,
                        1.0,
                        1.0
                        };

        double[][] aval = {
            new double[] {1.0},

```

(continues on next page)

```

        new double[] {1.0},
        new double[] {2.0}
};

int[] [] asub = {
    new int[] {0},
    new int[] {0},
    new int[] {0}
};

int[] csub = new int[3];

// Make mosek environment.
using (mosek.Env env = new mosek.Env())
{
    // Create a task object.
    using (mosek.Task task = new mosek.Task(env, 0, 0))
    {
        // Directs the log task stream to the user specified
        // method msgclass.streamCB
        task.set_Stream (mosek.streamtype.log, new msgclass (""));

        /* 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 (int j = 0; j < numvar; ++j)
        {
            /* 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, bkc[j], blx[j], bux[j]);
        }

        for (int j = 0; j < aval.Length; ++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.
           for i=1, ...,numcon : blc[i] <= constraint i <= buc[i] */
        for (int i = 0; i < numcon; ++i)
            task.putconbound(i, bkc[i], blc[i], buc[i]);

        csub[0] = 3;
        csub[1] = 0;
        csub[2] = 1;
        task.appendcone(mosek.conetype.quad,
                        0.0, /* For future use only, can be set to 0.0 */
                        csub);
    }
}

```

(continues on next page)

```

    csub[0] = 4;
    csub[1] = 5;
    csub[2] = 2;
    task.appendcone(mosek.conetype.rquad, 0.0, csub);

    task.putobjsense(mosek.objsense.minimize);
    task.optimize();
    // Print a summary containing information
    // about the solution for debugging purposes
    task.solutionsummary(mosek.streamtype.msg);

    mosek.solsta solsta;
    /* Get status information about the solution */
    task.getsolsta(mosek.soltype.itr, out solsta);

    double[] xx = new double[numvar];

    task.getxx(mosek.soltype.itr, // Basic solution.
              xx);

    switch (solsta)
    {
        case mosek.solsta.optimal:
            Console.WriteLine ("Optimal primal solution\n");
            for (int j = 0; j < numvar; ++j)
                Console.WriteLine ("x[{0}]: {1}", j, xx[j]);
            break;
        case mosek.solsta.dual_infeas_cer:
        case mosek.solsta.prim_infeas_cer:
            Console.WriteLine("Primal or dual infeasibility.\n");
            break;
        case mosek.solsta.unknown:
            Console.WriteLine("Unknown solution status.\n");
            break;
        default:
            Console.WriteLine("Other solution status");
            break;
    }
}
}
}
}
}
}
}

```



## 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

A cone is defined using the function `Task.appendcone`:

```
task.appendcone(mosek.conetype.ppow, 0.2, new int[3] {0, 1, 3});
task.appendcone(mosek.conetype.ppow, 0.4, new int[3] {2, 5, 4});
```

The first argument selects the type of power cone, that is `conetype.ppow`. The second argument is the cone parameter  $\alpha$ . The remaining arguments list the variables which form the cone. Variants of this method are available to append multiple cones at a time.

The code below produces the answer of (6.7) which is

```
[ 0.06389298  0.78308564  2.30604283 ]
```

## Source code

Listing 6.5: Source code solving problem (6.7).

```
using System;

namespace mosek.example
{
    class msgclass : mosek.Stream
    {
        string prefix;
        public msgclass (string prfx)
        {
            prefix = prfx;
        }

        public override void streamCB (string msg)
        {
            Console.Write ("{0}{1}", prefix, msg);
        }
    }

    public class ceo1
    {
        public static void Main ()
        {
            const int numcon = 1;
            const int numvar = 6;

            // Since the value infinity is never used, we define
            // 'infinity' symbolic purposes only
            double infinity = 0;

            mosek.boundkey[] bkey = new mosek.boundkey[numvar];
            double[] blx = new double[numvar];
            double[] bux = new double[numvar];

            double[] val = { 1.0, 1.0, -1.0 };
        }
    }
}
```

(continues on next page)

```

int[]    sub    = { 3, 4, 0 };

double[] aval   = { 1.0, 1.0, 0.5 };
int[]    asub   = { 0, 1, 2 };

int i;
double[] xx     = new double[numvar];

// Make mosek environment.
using (mosek.Env env = new mosek.Env())
{
    // Create a task object.
    using (mosek.Task task = new mosek.Task(env, 0, 0))
    {
        // Directs the log task stream to the user specified
        // method msgclass.streamCB
        task.set_Stream (mosek.streamtype.log, new msgclass (""));

        /* 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(sub, val);
        task.putarow(0, asub, aval);
        task.putconbound(0, mosek.boundkey.fx, 2.0, 2.0);
        for(i=0;i<5;i++) {
            bkg[i] = mosek.boundkey.fr;
            blx[i] = -infinity;
            bux[i] = infinity;
        }
        bkg[5] = mosek.boundkey.fx;
        blx[5] = bux[5] = 1.0;
        task.putvarboundslice(0, numvar, bkg, blx, bux);

        /* Add a conic constraint */
        task.appendcone(mosek.conetype.ppow, 0.2, new int[3] {0, 1, 3});
        task.appendcone(mosek.conetype.ppow, 0.4, new int[3] {2, 5, 4});

        task.putobjsense(mosek.objsense.maximize);
        task.optimize();

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

        mosek.solsta solsta;
        /* Get status information about the solution */
        task.getsolsta(mosek.soltype.itr, out solsta);

        task.getxx(mosek.soltype.itr, // Basic solution.
                   xx);
    }
}

```





```

class msgclass : mosek.Stream
{
    string prefix;
    public msgclass (string prfx)
    {
        prefix = prfx;
    }

    public override void streamCB (string msg)
    {
        Console.Write ("{0}{1}", prefix, msg);
    }
}

public class ceo1
{
    public static void Main ()
    {
        const int numcon = 1;
        const int numvar = 3;

        // Since the value infinity is never used, we define
        // 'infinity' symbolic purposes only
        double infinity = 0;

        mosek.boundkey bkc = mosek.boundkey.fx;
        double blc = 1.0 ;
        double buc = 1.0 ;

        mosek.boundkey[] bkc = {mosek.boundkey.fr,
                                mosek.boundkey.fr,
                                mosek.boundkey.fr
                                };
        double[] blx = { -infinity,
                        -infinity,
                        -infinity
                        };
        double[] bux = { +infinity,
                        +infinity,
                        +infinity
                        };

        double[] c    = { 1.0,
                          1.0,
                          0.0
                          };

        double[] a = { 1.0, 1.0, 1.0 };
        int[] asub = { 0, 1, 2 };
        int[] csub = new int[3];

        // Make mosek environment.
        using (mosek.Env env = new mosek.Env())
        {
            // Create a task object.
            using (mosek.Task task = new mosek.Task(env, 0, 0))

```

```

{
    // Directs the log task stream to the user specified
    // method msgclass.streamCB
    task.set_Stream (mosek.streamtype.log, new msgclass (""));

    /* 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, asub, a);
    task.putconbound(0, bkc, blc, buc);
    task.putvarboundslice(0, numvar, bkx, blx, bux);

    /* Define the exponential cone */
    csub[0] = 0;
    csub[1] = 1;
    csub[2] = 2;
    task.appendcone(mosek.conetype.pexp,
                   0.0, /* For future use only, can be set to 0.0 */
                   csub);

    task.putobjsense(mosek.objsense.minimize);
    task.optimize();

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

    mosek.solsta solsta;
    /* Get status information about the solution */
    task.getsolsta(mosek.soltype.itr, out solsta);

    double[] xx = new double[numvar];

    task.getxx(mosek.soltype.itr, // Basic solution.
              xx);

    switch (solsta)
    {
    case mosek.solsta.optimal:
        Console.WriteLine ("Optimal primal solution\n");
        for (int j = 0; j < numvar; ++j)
            Console.WriteLine ("x[{0}]: {1}", j, xx[j]);
        break;
    case mosek.solsta.dual_infeas_cer:
    case mosek.solsta.prim_infeas_cer:
        Console.WriteLine("Primal or dual infeasibility.\n");
        break;
    case mosek.solsta.unknown:
        Console.WriteLine("Unknown solution status.\n");
    }
}

```





## Source code

Listing 6.7: Source code solving problem (6.11).

```
using System;

namespace mosek.example
{
    public class sdo1
    {
        public static void Main(string[] args)
        {
            int    numcon    = 2; /* Number of constraints. */
            int    numvar    = 3; /* Number of conic quadratic variables */
            int[]  dimbarvar = { 3 }; /* Dimensions of semidefinite cones */
            int[]  lenbarvar = { 3 * (3 + 1) / 2 }; /* Number of scalar SD variables */

            mosek.boundkey[] bkc = { mosek.boundkey.fx, mosek.boundkey.fx };
            double[]      blc   = { 1.0, 0.5 };
            double[]      buc   = { 1.0, 0.5 };

            int[]      barc_i = { 0, 1, 1, 2, 2 },
                    barc_j = { 0, 0, 1, 1, 2 };
            double[]   barc_v = { 2.0, 1.0, 2.0, 1.0, 2.0 };

            int[] []   asub    = { new int[] {0}, new int[] {1, 2}}; /* column
↳subscripts of A */
            double[] [] aval    = { new double[] {1.0}, new double[] {1.0, 1.0}};

            int[] []   bara_i = { new int[] {0, 1, 2}, new int[] {0, 1, 2, 1,
↳2, 2 } },
                    bara_j = { new int[] {0, 1, 2}, new int[] {0, 0, 0, 1,
↳1, 2 } };
            double[] [] bara_v = { new double[] {1.0, 1.0, 1.0}, new double[] {1.0, 1.0,
↳1.0, 1.0, 1.0, 1.0}};
            int[]      conesub = { 0, 1, 2 };

            using (mosek.Env env = new mosek.Env())
            {
                // Create a task object.
                using (mosek.Task task = new mosek.Task(env, 0, 0))
                {
                    // Directs the log task stream to the user specified
                    // method msgclass.streamCB
                    task.set_Stream (mosek.streamtype.log, new msgclass (""));
                    /* 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);

                    /* Append 'NUMBARVAR' semidefinite variables. */
                    task.appendbarvars(dimbarvar);
                }
            }
        }
    }
}
```

(continues on next page)

```

/* Optionally add a constant term to the objective. */
task.putcfix(0.0);

/* Set the linear term c_j in the objective.*/
task.putcj(0, 1.0);

for (int j = 0; j < numvar; ++j)
    task.putvarbound(j, mosek.boundkey.fr, -0.0, 0.0);

/* Set the linear term barc_j in the objective.*/
{
    long[] idx = new long[1];
    double[] falpha = { 1.0 };
    idx[0] = task.appendsparsesymmat(dimbarvar[0],
                                     barc_i,
                                     barc_j,
                                     barc_v);

    task.putbarcj(0, idx, falpha);
}

/* Set the bounds on constraints.
   for i=1, ..., numcon : blc[i] <= constraint i <= buc[i] */

for (int i = 0; i < numcon; ++i)
    task.putconbound(i,          /* Index of constraint.*/
                     bkc[i],     /* Bound key.*/
                     blc[i],     /* Numerical value of lower bound.*/
                     buc[i]);    /* Numerical value of upper bound.*/

/* Input A row by row */
for (int i = 0; i < numcon; ++i)
    task.putarow(i,
                 asub[i],
                 aval[i]);

/* Append the conic quadratic cone */
task.appendcone(mosek.conetype.quad,
               0.0,
               conesub);

/* Add the first row of barA */
{
    long[] idx = new long[1];
    double[] falpha = {1.0};
    task.appendsparsesymmat(dimbarvar[0],
                             bara_i[0],
                             bara_j[0],
                             bara_v[0],
                             out idx[0]);

    task.putbaraij(0, 0, idx, falpha);
}

{
    long[] idx = new long[1];

```

(continues on next page)

```

double[] falpha = {1.0};
/* Add the second row of barA */
task.appendsparsesymmat(dimbarvar[0],
                        bara_i[1],
                        bara_j[1],
                        bara_v[1],
                        out idx[0]);

task.putbaraij(1, 0, idx, falpha);
}

/* Run optimizer */
task.optimize();

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

mosek.solsta solsta;
task.getsolsta (mosek.soltype.itr, out solsta);

switch (solsta)
{
    case mosek.solsta.optimal:
        double[] xx = new double[numvar];
        double[] barx = new double[lenbarvar[0]];

        task.getxx(mosek.soltype.itr, xx);
        task.getbarxj(mosek.soltype.itr, /* Request the interior solution. */
                     0,
                     barx);
        Console.WriteLine("Optimal primal solution");
        for (int i = 0; i < numvar; ++i)
            Console.WriteLine("x[{0}] : {1}", i, xx[i]);

        for (int i = 0; i < lenbarvar[0]; ++i)
            Console.WriteLine("barx[{0}]: {1}", i, barx[i]);
        break;
    case mosek.solsta.dual_infeas_cer:
    case mosek.solsta.prim_infeas_cer:
        Console.WriteLine("Primal or dual infeasibility certificate found.");
        break;
    case mosek.solsta.unknown:
        Console.WriteLine("The status of the solution could not be determined.
↪");
        break;
    default:
        Console.WriteLine("Other solution status.");
        break;
}
}
}
}
}
}

```



```

using (mosek.Task task = new mosek.Task(env, 0, 0))
{
    // Directs the log task stream to the user specified
    // method task_msg_obj.stream
    task.set_Stream (mosek.streamtype.log, new msgclass (""));

    /* Append numcon empty constraints.
       The constraints will initially have no bounds. */
    task.appendcons(numcon);

    /* Append numbarvar semidefinite variables. */
    task.appendbarvars(dimbarvar);

    /* Set objective (6 nonzeros).*/
    task.putbarblocktriplet(6, Cj, Ck, Cl, Cv);

    /* Set the equality constraint (6 nonzeros).*/
    task.putbarablocktriplet(6, Ai, Aj, Ak, Al, Av);

    /* Set the inequality constraint (1 nonzero).*/
    task.putbarablocktriplet(1, A2i, A2j, A2k, A2l, A2v);

    /* Set constraint bounds */
    task.putconboundslice(0, 2, bkc, blc, buc);

    /* Run optimizer */
    task.optimize();
    task.solutionsummary(mosek.streamtype.msg);

    mosek.solsta solsta = task.getsolsta(mosek.soltype.itr);

    switch (solsta) {
        case mosek.solsta.optimal:

            /* Retrieve the solution for all symmetric variables */
            Console.WriteLine("Solution (lower triangular part vectorized):");
            for(int i = 0; i < numbarvar; i++) {
                int dim = dimbarvar[i] * (dimbarvar[i] + 1) / 2;
                double[] barx = new double[dim];

                task.getbarxj(mosek.soltype.itr, i, barx);

                Console.Write("X" + (i+1) + ": ");
                for (int j = 0; j < dim; ++j)
                    Console.Write(barx[j] + " ");
                Console.WriteLine();
            }

            break;
        case mosek.solsta.dual_infeas_cer:
        case mosek.solsta.prim_infeas_cer:
            Console.WriteLine("Primal or dual infeasibility certificate found.");
            break;
        case mosek.solsta.unknown:
            Console.WriteLine("The status of the solution could not be determined.
→");

```

(continues on next page)



```

    {
        Console.Write ("{0}", msg);
    }
}

public class milo1
{
    public static void Main ()
    {
        const int numcon = 2;
        const int numvar = 2;

        // Since the value infinity is never used, we define
        // 'infinity' symbolic purposes only
        double infinity = 0;

        mosek.boundkey[] bkc = { mosek.boundkey.up,
                                mosek.boundkey.lo
                                };
        double[] blc = { -infinity,
                        -4.0
                        };
        double[] buc = { 250.0,
                        infinity
                        };

        mosek.boundkey[] bkc = { mosek.boundkey.lo,
                                mosek.boundkey.lo
                                };
        double[] blx = { 0.0,
                        0.0
                        };
        double[] bux = { infinity,
                        infinity
                        };

        double[] c = {1.0, 0.64 };
        int[][] asub = { new int[] {0, 1}, new int[] {0, 1} };
        double[][] aval = { new double[] {50.0, 3.0}, new double[] {31.0, -2.0} };

        double[] xx = new double[numvar];

        mosek.Env env = null;
        mosek.Task task = null;

        try
        {
            // Make mosek environment.
            env = new mosek.Env ();
            // Create a task object linked with the environment env.
            task = new mosek.Task (env, numcon, numvar);
            // Directs the log task stream to the user specified
            // method task_msg_obj.streamCB
            MsgClass task_msg_obj = new MsgClass ();
            task.set_Stream (mosek.streamtype.log, task_msg_obj);
        }
    }
}

```

(continues on next page)

```

/* Give MOSEK an estimate of the size of the input data.
   This is done to increase the speed of inputting data.
   However, it is optional. */
/* 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 (int j = 0; j < numvar; ++j)
{
    /* 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, bkg[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.
   for i=1, ..., numcon : blc[i] <= constraint i <= buc[i] */
for (int i = 0; i < numcon; ++i)
    task.putconbound(i, bkc[i], blc[i], buc[i]);

/* Specify integer variables. */
for (int j = 0; j < numvar; ++j)
    task.putvartype(j, mosek.variabletype.type_int);
task.putobjsense(mosek.objsense.maximize);

/* Set max solution time */
task.putdoupam(mosek.dparam.mio_max_time, 60.0);

task.optimize();

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

mosek.solsta solsta;
/* Get status information about the solution */
task.getsolsta(mosek.soltype.itg, out solsta);
task.getxx(mosek.soltype.itg, // Integer solution.
           xx);

switch (solsta)
{

```

(continues on next page)

```

    case mosek.solsta.optimal:
        Console.WriteLine ("Optimal primal solution\n");
        for (int j = 0; j < numvar; ++j)
            Console.WriteLine ("x[{0}]:", xx[j]);
        break;
    case mosek.solsta.prim_feas:
        Console.WriteLine ("Feasible primal solution\n");
        for (int j = 0; j < numvar; ++j)
            Console.WriteLine ("x[{0}]:", xx[j]);
        break;
    case mosek.solsta.unknown:
        mosek.prosta prosta;
        task.getprosta(mosek.soltype.itg, out prosta);
        switch (prosta)
        {
            case mosek.prosta.prim_infeas_or_unbounded:
                Console.WriteLine("Problem status Infeasible or unbounded");
                break;
            case mosek.prosta.prim_infeas:
                Console.WriteLine("Problem status Infeasible.");
                break;
            case mosek.prosta.unknown:
                Console.WriteLine("Problem status unknown.");
                break;
            default:
                Console.WriteLine("Other problem status.");
                break;
        }
        break;
    default:
        Console.WriteLine("Other solution status");
        break;
}
}
catch (mosek.Exception e)
{
    Console.WriteLine (e.Code);
    Console.WriteLine (e);
    throw;
}
finally
{
    if (task != null) task.Dispose ();
    if (env != null) env.Dispose ();
}
}
}
}

```



Listing 6.12: Implementation of problem (6.16).

```

public class mico1
{
    public static void Main ()
    {
        mosek.Env env = new mosek.Env ();
        mosek.Task task = new mosek.Task(env, 0, 0);

        // Directs the log task stream to the user specified
        // method task_msg_obj.streamCB
        MsgClass task_msg_obj = new MsgClass ();
        task.set_Stream (mosek.streamtype.log, task_msg_obj);

        task.appendvars(6);
        task.appendcons(3);
        task.putvarboundsliceconst(0, 6, mosek.boundkey.fr, -0.0, 0.0);

        // Integrality constraints
        task.putvartypelist(new int[]{1,2},
                           new mosek.variabletype[]{mosek.variabletype.type_int, mosek.
→variabletype.type_int});

        // Set up the three auxiliary linear constraints
        task.putaijlist(new int[]{0,0,1,2,2},
                       new int[]{1,3,4,2,5},
                       new double[]{-1,1,1,1,-1});
        task.putconboundslice(0, 3,
                               new mosek.boundkey[]{mosek.boundkey.fx, mosek.boundkey.fx,
→ mosek.boundkey.fx},
                               new double[]{-3.8, 1, 0}, new double[]{-3.8, 1, 0});

        // Objective
        task.putobjsense(mosek.objsense.minimize);
        task.putcj(0, 1);

        // Conic part of the problem
        task.appendconeseq(mosek.conetype.quad, 0, 3, 0);
        task.appendconeseq(mosek.conetype.pexp, 0, 3, 3);

        // Optimize the task
        task.optimize();
        task.solutionsummary(mosek.streamtype.msg);

        double[] xx = {0, 0};
        task.getxxslice(mosek.soltype.itg, 1, 3, xx);
        Console.WriteLine ("x = {0}, y = {1}", xx[0], xx[1]);
    }
}

```

Error and solution status handling were omitted for readability.



Listing 6.13: Implementation of log-sum-exp as in (6.21).

```

// Add a single log-sum-exp constraint  $\sum(\log(\exp(z_i))) \leq 0$ 
// Assume numExp variable triples are ordered as (u0,t0,z0,u1,t1,z1...)
// starting from variable with index expStart
double[] val = new double[numExp];
int[] sub = new int[numExp];

//  $\sum(u_i) = 1$  as constraint number c,  $u_i$  unbounded
for(int i = 0; i < numExp; i++)
    { sub[i] = expStart + 3*i; val[i] = 1.0; }
task.putarow(c, sub, val);
task.putconbound(c, boundkey.fx, 1.0, 1.0);
task.putvarboundlistconst(sub, boundkey.fr, -inf, inf);

//  $z_i$  unbounded
for(int i = 0; i < numExp; i++) sub[i] = expStart + 3*i + 2;
task.putvarboundlistconst(sub, boundkey.fr, -inf, inf);

//  $t_i = 1$ 
for(int i = 0; i < numExp; i++) sub[i] = expStart + 3*i + 1;
task.putvarboundlistconst(sub, boundkey.fx, 1, 1);

// Every triple is in an exponential cone
conetype[] ct = new conetype[numExp];
int[] len = new int[numExp];
for(int i = 0; i < numExp; i++)
    { ct[i] = conetype.pexp; val[i] = 0.0; len[i] = 3; }
task.appendconeseq(ct, val, len, expStart);

```

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 6.14: Source code solving problem (6.20).

```

public static double[] max_volume_box(double Aw, double Af,
                                     double alpha, double beta, double gamma,
↳double delta)
{
    // Basic dimensions of our problem
    int numvar = 3; // Variables in original problem
    int numLinCon = 3; // Linear constraints in original problem
    int numExp = 2; // Number of exp-terms in the log-sum-exp constraint

    // Linear part of the problem
    double[] cval = {1, 1, 1};
    int[] asubi = {0, 0, 1, 1, 2, 2};
    int[] asubj = {1, 2, 0, 1, 2, 1};
    double[] aval = {1.0, 1.0, 1.0, -1.0, 1.0, -1.0};
    boundkey[] bkc = {boundkey.up, boundkey.ra, boundkey.ra};
    double[] blc = {-inf, Math.Log(alpha), Math.Log(gamma)};
    double[] buc = {Math.Log(Af), Math.Log(beta), Math.Log(delta)};

    // Linear part setting up slack variables
    // for the linear expressions appearing inside exps
    //  $x_5 - x - y = \log(2/A_{wall})$ 
    //  $x_8 - x - z = \log(2/A_{wall})$ 

```

(continues on next page)

```

// The slack indexes are convenient for defining exponential cones, see later
int[]    a2subi = {3, 3, 3, 4, 4, 4};
int[]    a2subj = {5, 0, 1, 8, 0, 2};
double[] a2val  = {1.0, -1.0, -1.0, 1.0, -1.0, -1.0};
boundkey[] b2kc = {boundkey.fx, boundkey.fx};
double[] b2luc  = {Math.Log(2/Aw), Math.Log(2/Aw)};

using (Env env = new Env())
{
    using (Task task = new Task(env, 0, 0))
    {
        // Directs the log task stream to the user specified
        // method task_msg_obj.stream
        task.set_Stream (mosek.streamtype.log, new msgclass (""));

        // Add variables and constraints
        task.appendvars(numvar + 3*numExp);
        task.appendcons(numLinCon + numExp + 1);

        // 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);

        // Add linear constraints for the expressions appearing in exp(...)
        task.putaijlist(a2subi, a2subj, a2val);
        task.putconboundslice(numLinCon, numLinCon+numExp, b2kc, b2luc, b2luc);

        int c = numLinCon + numExp;
        int expStart = numvar;
        // Add a single log-sum-exp constraint  $\sum(\log(\exp(z_i))) \leq 0$ 
        // Assume numExp variable triples are ordered as (u0,t0,z0,u1,t1,z1...)
        // starting from variable with index expStart
        double[] val = new double[numExp];
        int[] sub = new int[numExp];

        //  $\sum(u_i) = 1$  as constraint number c,  $u_i$  unbounded
        for(int i = 0; i < numExp; i++)
        { sub[i] = expStart + 3*i; val[i] = 1.0; }
        task.putarow(c, sub, val);
        task.putconbound(c, boundkey.fx, 1.0, 1.0);
        task.putvarboundlistconst(sub, boundkey.fr, -inf, inf);

        //  $z_i$  unbounded
        for(int i = 0; i < numExp; i++) sub[i] = expStart + 3*i + 2;
        task.putvarboundlistconst(sub, boundkey.fr, -inf, inf);

        //  $t_i = 1$ 
        for(int i = 0; i < numExp; i++) sub[i] = expStart + 3*i + 1;
        task.putvarboundlistconst(sub, boundkey.fx, 1, 1);

        // Every triple is in an exponential cone
    }
}

```

(continued from previous page)

```
conetype[] ct = new conetype[numExp];
int[] len = new int[numExp];
for(int i = 0; i < numExp; i++)
    { ct[i] = conetype.pexp; val[i] = 0.0; len[i] = 3; }
task.appendconeseq(ct, val, len, expStart);

// Solve and map to original h, w, d
task.optimize();
double[] xyz = new double[numvar];
double[] hwd = new double[numvar];
task.getxxslice(soltype.itr, 0, numvar, xyz);
for(int i = 0; i < numvar; i++) hwd[i] = Math.Exp(xyz[i]);
return hwd;
}
}
```

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

Listing 6.15: Sample data for problem (6.19).

```
public static void Main(String[] args)
{
    double Aw    = 200.0;
    double Af    = 50.0;
    double alpha = 2.0;
    double beta  = 10.0;
    double gamma = 2.0;
    double delta = 10.0;

    double[] hwd = max_volume_box(Aw, Af, alpha, beta, gamma, delta);

    Console.WriteLine("h={0:f4} w={1:f4} d={2:f4}", hwd[0], hwd[1], hwd[2]);
}
```

## 6.9 Library of basic functions

This section contains a library of small models of basic functions frequently appearing in optimization models. It is essentially an implementation of the mathematical models from the [MOSEK Modeling Cookbook](#) using Optimizer API for .NET. These short code snippets can be seen as illustrative examples, can be copy-pasted to other code, and can even be directly called when assembling optimization models as we show in [Sec. 6.9.6](#) (although this may be more suitable for prototyping; also note that additional variables and constraints will be introduced and there is no error checking).

### 6.9.1 Variable and constraint management

#### Append variables

Adds a number of new variables. Returns the index of the first variable in the sequence.

Listing 6.16: New variables.

```
public static int msk_newvar(Task task, int num) { // free
    int v = task.getnumvar();
    task.appendvars(num);
    for(int i=0; i<num; i++)
        task.putvarbound(v+i, boundkey.fr, -inf, inf);
}
```

(continues on next page)

(continued from previous page)

```
    return v;
}
public static int msk_newvar_fx(Task task, int num, double val) { // fixed
    int v = task.getnumvar();
    task.appendvars(num);
    for(int i=0; i<num; i++)
        task.putvarbound(v+i, boundkey.fx, val, val);
    return v;
}
public static int msk_newvar_bin(Task task, int num) { // binary
    int v = task.getnumvar();
    task.appendvars(num);
    for(int i=0; i<num; i++) {
        task.putvarbound(v+i, boundkey.ra, 0.0, 1.0);
        task.putvartype(v+i, variabletype.type_int);
    }
    return v;
}
```

### Variable duplication

Declares equality of two variables, or returns an index of a new duplicate of an existing variable.

Listing 6.17: Duplicate variables.

```
// x = y
public static void msk_equal(Task task, int x, int y) {
    int c = msk_newcon(task, 1);
    task.putaij(c, x, 1.0);
    task.putaij(c, y, -1.0);
    task.putconbound(c, boundkey.fx, 0.0, 0.0);
}
public static int msk_dup(Task task, int x) {
    int y = msk_newvar(task, 1);
    msk_equal(task, x, y);
    return y;
}
```

### Append constraints

Adds a number of new constraints. Returns the index of the first constraint in the sequence.

Listing 6.18: New constraints.

```
public static int msk_newcon(Task task, int num) {
    int c = task.getnumcon();
    task.appendcons(num);
    return c;
}
```

## 6.9.2 Linear operations

### Absolute value

$$t \geq |x|$$

Listing 6.19: Absolute value.

```
// t >= |x|
public static void msk_abs(Task task, int t, int x) {
    int c = msk_newcon(task, 2);
    task.putaij(c, t, 1.0);
    task.putaij(c, x, 1.0);
    task.putconbound(c, boundkey.lo, 0.0, inf);
    task.putaij(c+1, t, 1.0);
    task.putaij(c+1, x, -1.0);
    task.putconbound(c+1, boundkey.lo, 0.0, inf);
}
```

### 1-norm

$$t \geq \sum_i |x_i|$$

Listing 6.20: 1-norm.

```
// t >= sum( |x_i| ), x is a list of variables
public static void msk_norm1(Task task, int t, int[] x) {
    int n = x.Length;
    int u = msk_newvar(task, n);
    for(int i=0; i<n; i++) msk_abs(task, u+i, x[i]);
    int c = msk_newcon(task, 1);
    for(int i=0; i<n; i++) task.putaij(c, u+i, -1.0);
    task.putaij(c, t, 1.0);
    task.putconbound(c, boundkey.lo, 0.0, inf);
}
```

## 6.9.3 Quadratic and power operations

### Square

$$t \geq x^2$$

Listing 6.21: Square.

```
// t >= x^2
public static void msk_sq(Task task, int t, int x) {
    task.appendcone(conetype.rquad, 0.0, new int[]{msk_newvar_fx(task, 1, 0.5), t,
↪x});
}
```

## 2-norm

$$t \geq \sqrt{\sum_i x_i^2}$$

Listing 6.22: 2-norm.

```
// t >= sqrt(x_1^2 + ... + x_n^2) where x is a list of variables
public static void msk_norm2(Task task, int t, int[] x) {
    int[] submem = new int[x.Length+1];
    submem[0] = t;
    for(int i=0; i<x.Length; i++) submem[i+1] = x[i];
    task.appendcone(conetype.quad, 0.0, submem);
}
```

## Powers

$$t \geq |x|^p, p > 1$$

Listing 6.23: Power.

```
// t >= |x|^p (where p>1)
public static void msk_pow(Task task, int t, int x, double p) {
    task.appendcone(conetype.ppow, 1.0/p, new int[]{t, msk_newvar_fx(task, 1, 1.0),
↪x});
}
```

$$t \geq 1/x^p, x > 0, p > 0$$

Listing 6.24: Power reciprocal.

```
// t >= 1/x^p, x>0 (where p>0)
public static void msk_pow_inv(Task task, int t, int x, double p) {
    task.appendcone(conetype.ppow, 1.0/(1.0+p), new int[]{t, x, msk_newvar_fx(task,
↪1, 1.0)});
}
```

## p-norm

$$t \geq (\sum_i |x_i|^p)^{1/p}, p > 1$$

Listing 6.25:  $p$ -norm.

```
// t >= ||x||_p (where p>1), x is a list of variables
public static void msk_pnorm(Task task, int t, int[] x, double p) {
    int n = x.Length;
    int r = msk_newvar(task, n);
    for(int i=0; i<n; i++)
        task.appendcone(conetype.ppow, 1.0-1.0/p, new int[]{t, r+i, x[i]});
    int c = msk_newcon(task, 1);
    for(int i=0; i<n; i++)
        task.putaij(c, r+i, -1.0);
    task.putaij(c, t, 1.0);
    task.putconbound(c, boundkey.fx, 0.0, 0.0);
}
```

### Geometric mean

$$t \leq (x_1 \cdots x_n)^{1/n}, x_i > 0$$

Listing 6.26: Geometric mean.

```
// |t| <= (x_1...x_n)^(1/n), x_i>=0, x is a list of variables of Length >= 1
public static void msk_geo_mean(Task task, int t, int n, int[] x) {
    if (n==1) msk_abs(task, x[0], t);
    else {
        int t2 = msk_newvar(task, 1);
        task.appendcone(conetype.ppow, 1.0-1.0/n, new int[]{t2, x[n-1], t});
        msk_geo_mean(task, msk_dup(task, t2), n-1, x);
    }
}
```

## 6.9.4 Exponentials and logarithms

### log

$$t \leq \log x, x > 0$$

Listing 6.27: Logarithm.

```
// t <= log(x), x>=0
public static void msk_log(Task task, int t, int x) {
    task.appendcone(conetype.pexp, 0.0, new int[]{x, msk_newvar_fx(task, 1, 1.0), t});
    ↪);
}
```

### exp

$$t \geq e^x$$

Listing 6.28: Exponential.

```
// t >= exp(x)
public static void msk_exp(Task task, int t, int x) {
    task.appendcone(conetype.pexp, 0.0, new int[]{t, msk_newvar_fx(task, 1, 1.0), x});
    ↪);
}
```

## Entropy

$$t \geq x \log x, \quad x > 0$$

Listing 6.29: Entropy.

```
// t >= x * log(x), x>=0
public static void msk_ent(Task task, int t, int x) {
    int v = msk_newvar(task, 1);
    int c = msk_newcon(task, 1);
    task.putaij(c, v, 1.0);
    task.putaij(c, t, 1.0);
    task.putconbound(c, boundkey.fx, 0.0, 0.0);
    task.appendcone(conetype.pexp, 0.0, new int[]{msk_newvar_fx(task, 1, 1.0), x, v}
↪);
}
```

## Relative entropy

$$t \geq x \log x/y, \quad x, y > 0$$

Listing 6.30: Relative entropy.

```
// t >= x * log(x/y), x,y>=0
public static void msk_relent(Task task, int t, int x, int y) {
    int v = msk_newvar(task, 1);
    int c = msk_newcon(task, 1);
    task.putaij(c, v, 1.0);
    task.putaij(c, t, 1.0);
    task.putconbound(c, boundkey.fx, 0.0, 0.0);
    task.appendcone(conetype.pexp, 0.0, new int[]{y, x, v});
}
```

## Log-sum-exp

$$\log \sum_i e^{x_i} \leq t$$

Listing 6.31: Log-sum-exp.

```
// log( sum_i(exp(x_i)) ) <= t, where x is a list of variables
public static void msk_logsumexp(Task task, int t, int[] x) {
    int n = x.Length;
    int u = msk_newvar(task, n);
    int z = msk_newvar(task, n);
    for(int i=0; i<n; i++) msk_exp(task, u+i, z+i);
    int c = msk_newcon(task, n);
    for(int i=0; i<n; i++) {
        task.putarow(c+i, new int[]{x[i], t, z+i}, new double[]{1.0, -1.0, -1.0});
        task.putconbound(c+i, boundkey.fx, 0.0, 0.0);
    }
    int s = msk_newcon(task, 1);
    for(int i=0; i<n; i++) task.putaij(s, u+i, 1.0);
    task.putconbound(s, boundkey.up, -inf, 1.0);
}
```

## 6.9.5 Integer Modeling

### Semicontinuous variable

$x \in \{0\} \cup [a, b]$ ,  $b > a > 0$

Listing 6.32: Semicontinuous variable.

```
// x = 0 or a <= x <= b
public static void msk_semicontinuous(Task task, int x, double a, double b) {
    int u = msk_newvar_bin(task, 1);
    int c = msk_newcon(task, 2);
    task.putarow(c, new int[]{x, u}, new double[]{1.0, -a});
    task.putconbound(c, boundkey.lo, 0.0, inf);
    task.putarow(c+1, new int[]{x, u}, new double[]{1.0, -b});
    task.putconbound(c+1, boundkey.up, -inf, 0.0);
}
```

### Indicator variable

$x \neq 0 \implies t = 1$ . We assume  $x$  is a priori normalized so  $|x_i| \leq 1$ .

Listing 6.33: Indicator variable.

```
// x!=0 implies t=1. Assumes that |x|<=1 in advance.
public static int msk_indicator(Task task, int x) {
    int t = msk_newvar_bin(task, 1);
    msk_abs(task, t, x);
    return t;
}
```

## Logical OR

At least one of the conditions is true.

Listing 6.34: Logical OR.

```
// x OR y, where x, y are binary
public static void msk_logic_or(Task task, int x, int y) {
    int c = msk_newcon(task, 1);
    task.putarow(c, new int[]{x, y}, new double[]{1.0, 1.0});
    task.putconbound(c, boundkey.lo, 1.0, inf);
}

// x_1 OR ... OR x_n, where x is sequence of variables
public static void msk_logic_or_vect(Task task, int[] x) {
    int c = msk_newcon(task, 1);
    for(int i=0; i<x.Length; i++) task.putaij(c, x[i], 1.0);
    task.putconbound(c, boundkey.lo, 1.0, inf);
}
```

## Logical NAND

At most one of the conditions is true (also known as SOS1).





and

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

Code in Listing 6.38 loads and solves this problem.

Listing 6.38: Setting up and solving problem (6.23)

```
// Since the value infinity is never used, we define  
// 'infinity' symbolic purposes only  
double  
infinity = 0;  
  
int numcon = 3;  
int numvar = 3;  
  
double[] c          = {1.5,  
                        2.5,  
                        3.0  
                        };  
mosek.boundkey[] bkc = {mosek.boundkey.up,  
                        mosek.boundkey.up,  
                        mosek.boundkey.up  
                        };  
double[] blc         = { -infinity,  
                        -infinity,  
                        -infinity  
                        };  
double[] buc         = {100000,  
                        50000,  
                        60000  
                        };  
mosek.boundkey[] bkc = {mosek.boundkey.lo,  
                        mosek.boundkey.lo,  
                        mosek.boundkey.lo  
                        };  
double[] blx         = {0.0,  
                        0.0,  
                        0.0  
                        };  
double[] bux         = { +infinity,  
                        +infinity,  
                        +infinity  
                        };  
  
int[][] asub = new int[numvar][];  
asub[0] = new int[] {0, 1, 2};  
asub[1] = new int[] {0, 1, 2};  
asub[2] = new int[] {0, 1, 2};  
  
double[][] aval = new double[numvar][];  
aval[0] = new double[] { 2.0, 3.0, 2.0 };  
aval[1] = new double[] { 4.0, 2.0, 3.0 };  
aval[2] = new double[] { 3.0, 3.0, 2.0 };  
  
double[] xx = new double[numvar];  
  
mosek.Task task = null;
```

(continues on next page)

```

mosek.Env env = null;

try
{
    // Create mosek environment.
    env = new mosek.Env ();
    // Create a task object linked with the environment env.
    task = new mosek.Task (env, numcon, numvar);

    /* Append the constraints. */
    task.appendcons(numcon);

    /* Append the variables. */
    task.appendvars(numvar);

    /* Put C. */
    task.putcfix(0.0);
    for (int j = 0; j < numvar; ++j)
        task.putcj(j, c[j]);

    /* Put constraint bounds. */
    for (int i = 0; i < numcon; ++i)
        task.putconbound(i, bkc[i], blc[i], buc[i]);

    /* Put variable bounds. */
    for (int j = 0; j < numvar; ++j)
        task.putvarbound(j, bkc[j], blx[j], bux[j]);

    /* Put A. */
    if ( numcon > 0 )
    {
        for (int j = 0; j < numvar; ++j)
            task.putacol(j,
                          asub[j],
                          aval[j]);
    }

    task.putobjsense(mosek.objsense.maximize);

    try
    {
        task.optimize();
    }
    catch (mosek.Warning w)
    {
        Console.WriteLine("Mosek warning:");
        Console.WriteLine (w.Code);
        Console.WriteLine (w);
    }

    task.getxx(mosek.soltype.bas, // Request the basic solution.
              xx);
}

```



and

$$x_0, x_1, x_2, x_3 \geq 0.$$

#### 6.10.4 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 6.40: Adding a new constraint.

```

/***** Add a new constraint *****/
/* Get index of new constraint */
int conidx;
task.getnumcon(out conidx);

/* Append a new constraint */
task.appendcons(1);
numcon++;

/* Set bounds on new constraint */
task.putconbound(conidx,
                  mosek.boundkey.up,
                  -infinity,
                  30000);

/* Put new values in the A matrix */
int[] arowsub = new int[] {0, 1, 2, 3};
double[] arowval = new double[] {1.0, 2.0, 1.0, 1.0};

task.putarow(conidx, /* row index */
              arowsub,
              arowval);

```

Again, we can continue with re-optimizing the modified problem.

#### 6.10.5 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



```

var n = tasks.Length;
var jobs = new System.Threading.Tasks.Task[n];

// Initialize
for (var i = 0; i < n; ++i)
{
    res[i] = mosek.rescode.err_unknown;
    trm[i] = mosek.rescode.err_unknown;
}

// Start parallel optimizations, one per task
for (var i = 0; i < n; ++i)
{
    int num = i;
    jobs[i] = System.Threading.Tasks.Task.Factory.StartNew( () => {
        try
        {
            trm[num] = tasks[num].optimize();
            res[num] = mosek.rescode.ok;
        }
        catch (mosek.Exception e)
        {
            trm[num] = mosek.rescode.err_unknown;
            res[num] = e.Code;
        }
    } );
}

// Join all threads
foreach (var j in jobs)
    j.Wait();
}

```

It remains to call the method with a few different tasks. When optimizing many task in parallel it usually makes sense to solve each task using one thread to avoid additional multitasking overhead. When all tasks complete we access the solutions in the standard way.

Listing 6.43: Calling the parallel optimizer.

```

/** Example of how to use ``paropt``.
Optimizes tasks whose names were read from command line.
*/
public static void Main(string[] argv)
{
    int n = argv.Length;
    mosek.Task[] t = new mosek.Task[n];
    mosek.rescode[] res = new mosek.rescode[n];
    mosek.rescode[] trm = new mosek.rescode[n];

    using (var env = new mosek.Env())
    {
        for(int i = 0; i < n; i++)
        {
            t[i] = new mosek.Task(env);
            t[i].readdata(argv[i]);
            // Each task will be single-threaded
            t[i].putintparam(mosek.iparam.intpnt_multi_thread, mosek.onoffkey.off);

```

(continues on next page)

```

    }

    paropt(t, res, trm);

    for(int i = 0; i < n; i++)
        Console.WriteLine("Task {0} res {1} trm {2} obj_val {3} time {4}",
            i,
            res[i],
            trm[i],
            t[i].getdouinf(mosek.dinfitem.intpnt_primal_obj),
            t[i].getdouinf(mosek.dinfitem.optimizer_time));
    }
}

```

Another, slightly more advanced application of the parallel optimizer is presented in [Sec. 11.3](#). For a more in-depth treatment see the following sections:

- [Case studies](#) for more advanced and complicated optimization examples.
- [Problem Formulation and Solutions](#) 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 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.

#### 7.1.1 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. 7.2](#) 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. 7.3](#).

If the optimization completes successfully, the next step is to check the solution status, as explained below.

#### 7.1.2 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	<i>soltype.bas</i>	<i>soltype.itr</i>	
Nonlinear continuous problem		<i>soltype.itr</i>	
Problem with integer variables			<i>soltype.itg</i>



Table 7.3: Integer problems (solution status for integer solution, others undefined)

Outcome	Problem status	Solution status
Integer optimal	<i>prosta.prim_feas</i>	<i>solsta.integer_optimal</i>
Infeasible	<i>prosta.prim_infeas</i>	<i>solsta.unknown</i>
Integer feasible point	<i>prosta.prim_feas</i>	<i>solsta.prim_feas</i>
No conclusion	<i>prosta.unknown</i>	<i>solsta.unknown</i>

### 7.1.4 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*.

### 7.1.5 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.

```
using System;
using mosek;
using System.Text;

namespace mosek.example
{
    // A log handler class
    class msgclass : mosek.Stream
    {
        public msgclass () {}
        public override void streamCB (string msg) { Console.Write ("{0}", msg); }
    }

    public class response
    {
        public static void Main(string[] argv)
        {
            StringBuilder symname = new StringBuilder();
            StringBuilder desc = new StringBuilder();

            string filename;
            if (argv.Length >= 1) filename = argv[0];
            else filename = "../data/cqo1.mps";

            // Define environment and task
            using (Env env = new Env())
            {
                using (Task task = new Task(env, 0, 0))
                {
                    try
```

(continues on next page)

```

{
    // (Optionally) set a log handler
    // task.set_Stream (streamtype.log, new msgclass ());

    // (Optionally) uncomment this to get solution status unknown
    // task.putintparam(iparam.intpnt_max_iterations, 1);

    // In this example we read data from a file
    task.readdata(filename);

    // Perform optimization
    rescode trm = task.optimize();
    task.solutionsummary(streamtype.log);

    // Handle solution status. We expect Optimal
    solsta solsta = task.getsolsta(soltype.itr);

    switch ( solsta )
    {
        case solsta.optimal:
            // Optimal solution. Print variable values
            Console.WriteLine("An optimal interior-point solution is located.");
            int numvar = task.getnumvar();
            double[] xx = new double[numvar];
            task.getxx(soltype.itr, xx);
            for(int i = 0; i < numvar; i++)
                Console.WriteLine("x[" + i + "] = " + xx[i]);
            break;

        case solsta.dual_infeas_cer:
            Console.WriteLine("Dual infeasibility certificate found.");
            break;

        case solsta.prim_infeas_cer:
            Console.WriteLine("Primal infeasibility certificate found.");
            break;

        case solsta.unknown:
            /* The solutions status is unknown. The termination code
               indicates why the optimizer terminated prematurely. */
            Console.WriteLine("The solution status is unknown.");
            Env.getcodedesc(trm, symname, desc);
            Console.WriteLine(" Termination code: {0} {1}", symname, desc);
            break;

        default:
            Console.WriteLine("An unexpected solution status " + solsta);
            break;
    }
}
catch (mosek.Error e)
{
    Console.WriteLine("Unexpected optimization error ({0}) {1}", e.Code, e.
↵Message);
}
}

```

```

    }
  }
}

```

## 7.2 Errors and exceptions

### Exceptions

Almost every function in Optimizer API for .NET 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 nonexistent 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. 7.1](#).

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.putdoupparam(mosek.dparam.intpnt_co_tol_rel_gap, -1.0e-7);
}
catch (mosek.Exception e) {
    mosek.rescode res = e.Code;
    Console.WriteLine("Response code {0}\nMessage      {1}", res, e.Message);
}

```

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. 7.3](#)). 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.

```
using System;
using mosek;
using System.Text;

namespace mosek.example
{
    // A log handler class
    class msgclass : mosek.Stream
    {
        public msgclass () {}
        public override void streamCB (string msg) { Console.Write ("{0}", msg); }
    }

    public class response
    {
        public static void Main(string[] argv)
        {
            StringBuilder symname = new StringBuilder();
            StringBuilder desc = new StringBuilder();

            string filename;
            if (argv.Length >= 1) filename = argv[0];
            else filename = "../data/cqo1.mps";

            // Define environment and task
            using (Env env = new Env())
            {
                using (Task task = new Task(env, 0, 0))
                {
                    try
                    {
                        // (Optionally) set a log handler
                        // task.set_Stream (streamtype.log, new msgclass ());

                        // (Optionally) uncomment this to get solution status unknown
                        // task.putintparam(iparam.intpnt_max_iterations, 1);

                        // In this example we read data from a file
                        task.readdata(filename);

                        // Perform optimization
                        rescode trm = task.optimize();
                        task.solutionsummary(streamtype.log);

                        // Handle solution status. We expect Optimal
                        solsta solsta = task.getsolsta(soltype.itr);

                        switch ( solsta )
                        {
                            case solsta.optimal:
                                // Optimal solution. Print variable values

```

(continues on next page)

```

        Console.WriteLine("An optimal interior-point solution is located.");
        int numvar = task.getnumvar();
        double[] xx = new double[numvar];
        task.getxx(soltype.itr, xx);
        for(int i = 0; i < numvar; i++)
            Console.WriteLine("x[" + i + "] = " + xx[i]);
        break;

    case solsta.dual_infeas_cer:
        Console.WriteLine("Dual infeasibility certificate found.");
        break;

    case solsta.prim_infeas_cer:
        Console.WriteLine("Primal infeasibility certificate found.");
        break;

    case solsta.unknown:
        /* The solutions status is unknown. The termination code
           indicates why the optimizer terminated prematurely. */
        Console.WriteLine("The solution status is unknown.");
        Env.getcodedesc(trm, symname, desc);
        Console.WriteLine("  Termination code: {0} {1}", symname, desc);
        break;

    default:
        Console.WriteLine("An unexpected solution status " + solsta);
        break;
    }
}
catch (mosek.Error e)
{
    Console.WriteLine("Unexpected optimization error ({0}) {1}", e.Code, e.
↵Message);
}
}
}
}
}
}
}

```

## 7.3 Input/Output

The logging and I/O features are provided mainly by the **MOSEK** task and to some extent by the **MOSEK** environment objects.



### 7.3.3 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 an OPF file with

```
task.writedata("data.opf");
```

All formats can be compressed with `gzip` by appending the `.gz` 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.

### 7.3.4 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 = new mosek.Task(env, 0, 0);
try {
    task.readdata("file.task.gz");
    task.optimize();
} catch (mosek.Exception) {
    Console.WriteLine("Problem reading the file");
}
```

## 7.4 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 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.putdoupparam*.
- 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 parameters which choose and tune the interior point optimizer 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.putdoupparam(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.putnadoupparam("MSK_DPAR_INTPNT_CO_TOL_REL_GAP", 1.0e-7 );

// Incorrect value
try
{
    task.putdoupparam(mosek.dparam.intpnt_co_tol_rel_gap, -1.0);
}
catch (mosek.Error)
{
    Console.WriteLine("Wrong parameter value");
}
```

## Reading parameter values

The functions *Task.getintparam*, *Task.getdoupparam*, *Task.getstrparam* can be used to inspect the current value of a parameter, for example:

```
double param = task.getdoupparam(mosek.dparam.intpnt_co_tol_rel_gap);
Console.WriteLine("Current value for parameter intpnt_co_tol_rel_gap = " +
    ↪param);
```

## 7.5 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. 7.6](#) 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.getdounf* 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.

```
double tm = task.getdounf(mosek.dinfitem.optimizer_time);
int iter = task.getintinf(mosek.iinfitem.intpnt_iter);

Console.WriteLine("Time: " + tm);
Console.WriteLine("Iterations: " + iter);
```

## 7.6 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.



## 7.6.4 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.

```
class myCallback : mosek.DataCallback
{
    double maxtime;

    public myCallback(double maxtime_)
    {
        maxtime = maxtime_;
    }

    public override int callback( callbackcode caller,
                                double[]    douinf,
                                int[]       intinf,
                                long[]      lintinf )
    {
        double opttime = 0.0;
        int itrn;
        double pobj, dobj, stime;

        switch (caller)
        {
            case callbackcode.begin_intpnt:
                Console.WriteLine("Starting interior-point optimizer");
                break;
            case callbackcode.intpnt:
                itrn    = intinf[(int) iinfitem.intpnt_iter    ];
                pobj    = douinf[(int) dinfitem.intpnt_primal_obj];
                dobj    = douinf[(int) dinfitem.intpnt_dual_obj ];
                stime    = douinf[(int) dinfitem.intpnt_time    ];
                opttime = douinf[(int) dinfitem.optimizer_time ];

                Console.WriteLine("Iterations: {0,-3}",itrn);
                Console.WriteLine(" Elapsed: Time: {0,6:F2}({1:F2})",opttime,stime);
                Console.WriteLine(" Primal obj.: {0,-18:E6} Dual obj.: {1,018:E6}e",pobj,
↪dobj);
                break;
            case callbackcode.end_intpnt:
                Console.WriteLine("Interior-point optimizer finished.");
                break;
            case callbackcode.begin_primal_simplex:
                Console.WriteLine("Primal simplex optimizer started.");
                break;
            case callbackcode.update_primal_simplex:
                itrn    = intinf[(int) iinfitem.sim_primal_iter ];
                pobj    = douinf[(int) dinfitem.sim_obj        ];
                stime    = douinf[(int) dinfitem.sim_time       ];
                opttime = douinf[(int) dinfitem.optimizer_time ];

                Console.WriteLine("Iterations: {0,-3}", itrn);
                Console.WriteLine(" Elapsed time: {0,6:F2}({1:F2})",opttime,stime);
                Console.WriteLine(" Obj.: {0,-18:E6}", pobj );
                break;
        }
    }
}
```

(continues on next page)

```

case callbackcode.end_primal_simplex:
    Console.WriteLine("Primal simplex optimizer finished.");
    break;
case callbackcode.begin_dual_simplex:
    Console.WriteLine("Dual simplex optimizer started.");
    break;
case callbackcode.update_dual_simplex:
    itrn    = intinf[(int) iinfitem.sim_dual_iter    ];
    pobj    = douinf[(int) dinfitem.sim_obj          ];
    stime   = douinf[(int) dinfitem.sim_time         ];
    opttime = douinf[(int) dinfitem.optimizer_time   ];
    Console.WriteLine("Iterations: {0,-3}", itrn);
    Console.WriteLine("  Elapsed time: {0,6:F2}({1:F2})",opttime,stime);
    Console.WriteLine("  Obj.: {0,-18:E6}", pobj );
    break;
case callbackcode.end_dual_simplex:
    Console.WriteLine("Dual simplex optimizer finished.");
    break;
case callbackcode.begin_bi:
    Console.WriteLine("Basis identification started.");
    break;
case callbackcode.end_bi:
    Console.WriteLine("Basis identification finished.");
    break;
default:
    break;
}

if (opttime >= maxtime)
    // mosek is spending too much time. Terminate it.
    return 1;

return 0;
}
}

```

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.

```
task.set_InfoCallback(new myCallback(maxtime));
```

## 7.7 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.

### 7.7.1 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.

```
using System;

namespace mosek.example
{
    class msgclass : mosek.Stream
    {
        public override void streamCB (string msg)
        {
            Console.Write ("{0}", msg);
        }
    }

    public class simple
    {
        public static void Main (string[] args)
        {
            if (args.Length < 2)
            {
                Console.WriteLine ("Missing arguments, syntax is:");
                Console.WriteLine (" opt_server_sync inputfile serveraddr");
            }
            else
            {
                String inputfile = args[0];
                String addr      = args[1];

                mosek.rescode trm;

                using (mosek.Env env = new mosek.Env())
                {
                    using (mosek.Task task = new mosek.Task(env))
                    {
```

(continues on next page)

(continued from previous page)

```
task.set_Stream (mosek.streamtype.log, new msgclass ());

task.readdata (inputfile);

// Set OptServer URL
task.putoptserverhost(addr);

// Optimize
task.optimize (out trm);

task.solutionsummary (mosek.streamtype.log);
    }
}
}
}
}
```

### 7.7.2 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.

```
using System;
using System.Threading;

namespace mosek.example
{
    class msgclass : mosek.Stream
    {
        public override void streamCB (string msg)
        {
            Console.Write ("{0}", msg);
        }
    }

    public class opt_server_async
    {
        public static void Main (string[] args)
        {
            if (args.Length == 0)
```

(continues on next page)

```

{
    Console.WriteLine ("Missing argument, syntax is:");
    Console.WriteLine ("  opt_server inputfile host port numpolls");
}
else
{
    string inputfile = args[0];
    string host      = args[1];
    string port      = args[2];
    int numpolls     = Convert.ToInt32(args[3]);

    using (mosek.Env env = new mosek.Env())
    {
        string token;

        using (mosek.Task task = new mosek.Task(env))
        {
            task.readdata (inputfile);
            token = task.asyncoptimize (host, port);
        }

        using (mosek.Task task = new mosek.Task(env))
        {
            task.readdata (inputfile);
            task.set_Stream (mosek.streamtype.log, new msgclass ());
            Console.WriteLine("Starting polling loop...");

            int i = 0;

            while ( true )
            {
                Thread.Sleep(500);
                Console.WriteLine("poll {0}...\n", i);

                mosek.rescode trm;
                mosek.rescode resp;

                int respavailable = task.asyncpoll( host,
                                                    port,
                                                    token,
                                                    out resp,
                                                    out trm);

                Console.WriteLine("polling done");

                if (respavailable != 0)
                {
                    Console.WriteLine("solution available!");

                    task.asyncgetresult(host,
                                        port,
                                        token,
                                        out resp,

```

```
        out trm);

        task.solutionsummary (mosek.streamtype.log);
        break;
    }

    if (i == numpolls)
    {
        Console.WriteLine("max num polls reached, stopping host.");
        task.asyncstop (host, port, token);
        break;
    }
    i++;
}
}
}
}
}
}
```

## 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. 7.3.1](#) for how to do it. In the simplest case:  
Create a log handler function:

```
class myStream : mosek.Stream
{
    public override void streamCB(string msg)
    {
        Console.WriteLine("{0}", msg);
    }
}
```

attach it to the log stream:

```
task.set_Stream(mosek.streamtype.log, new myStream());
```

and include solution summary after the call to optimize:

```
task.optimize();
task.solutionsummary(mosek.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. 7.3.3](#).
  - use a human-readable text format, such as \*.opf 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.opf");
```

- 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. 13.4.8](#).

### 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 : max
  Type           : CONIC (conic optimization problem)
  Constraints     : 20413
  Cones          : 2508
  Scalar variables : 20414
  Matrix variables : 0
  Integer variables : 0
```

This can be consulted to eliminate simple errors: wrong objective sense, wrong number of variables etc. Note that Fusion, and third-party modeling tools can introduce additional variables and constraints to the model. In the remaining **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

** Cones
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
```









### 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  $x \geq 5 \cdot 10^9$ .



















```

    prefix = prfx;
}

public override void streamCB (string msg)
{
    Console.Write ("{0}{1}", prefix, msg);
}
}

public class solvebasis
{
    public static void Main ()
    {
        const int numcon = 2;
        const int numvar = 2;

        // Since the value infinity is never used, we define
        // 'infinity' symbolic purposes only
        double
        infinity = 0;

        double[] c    = {1.0, 1.0};
        int[]    ptrb = {0, 2};
        int[]    ptre = {2, 3};
        int[]    asub = {0, 1,
                        0, 1
                        };
        double[] aval = {1.0, 1.0,
                        2.0, 1.0
                        };
        mosek.boundkey[] bkc = {mosek.boundkey.up,
                               mosek.boundkey.up
                               };

        double[] blc = { -infinity,
                        -infinity
                        };
        double[] buc = {2.0,
                        6.0
                        };

        mosek.boundkey[] bkc = {mosek.boundkey.lo,
                               mosek.boundkey.lo
                               };

        double[] blx = {0.0,
                        0.0
                        };

        double[] bux = { +infinity,
                        +infinity
                        };

        double[] w1 = {2.0, 6.0};
        double[] w2 = {1.0, 0.0};
        try
        {
            using (mosek.Env env = new mosek.Env())

```

(continues on next page)

```

{
    using (mosek.Task task = new mosek.Task(env))
    {
        task.set_Stream (mosek.streamtype.log, new msgclass ("[task]"));
        task.inputdata(numcon, numvar,
            c,
            0.0,
            ptrb,
            ptre,
            asub,
            aval,
            bkc,
            blc,
            buc,
            bkx,
            blx,
            bux);
        task.putobjsense(mosek.objsense.maximize);
        try
        {
            task.optimize();
        }
        catch (mosek.Warning w)
        {
            Console.WriteLine("Mosek warning:");
            Console.WriteLine (w.Code);
            Console.WriteLine (w);
        }

        int[] basis = new int[numcon];
        task.initbasissolve(basis);

        //List basis variables corresponding to columns of B
        int[] varsub = {0, 1};
        for (int i = 0; i < numcon; i++) {
            if (basis[varsub[i]] < numcon)
                Console.WriteLine ("Basis variable no {0} is xc{1}",
                    i,
                    basis[i]);
            else
                Console.WriteLine ("Basis variable no {0} is x{1}",
                    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.
        int nz = 2;

        task.solvewithbasis(0, ref nz, varsub, w1);
        Console.WriteLine ("nz = {0}", nz);
        Console.WriteLine ("Solution to Bx = w1:\n");

        for (int i = 0; i < nz; i++) {

```

(continues on next page)

```

        if (basis[varsub[i]] < numcon)
            Console.WriteLine ("xc {0} = {1}",
                                basis[varsub[i]],
                                w1[varsub[i]] );
        else
            Console.WriteLine ("x{0} = {1}",
                                basis[varsub[i]] - numcon,
                                w1[varsub[i]]);
    }

    // Solve B^Tx = w2
    nz = 1;
    varsub[0] = 0; // Only w2[0] is nonzero.

    task.solvewithbasis(1, ref nz, varsub, w2);

    Console.WriteLine ("\nSolution to B^Ty = w2:\n");

    for (int i = 0; i < nz; i++)
    {
        Console.WriteLine ("y {0} = {1}",
                            varsub[i],
                            w2[varsub[i]]);
    }
}
}
}
}
catch (mosek.Exception e)
{
    Console.WriteLine (e.Code);
    Console.WriteLine (e);
    throw;
}
}
}
}

```

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
xc1 = -4.000000e+00

Solution to B^Tx = c:

x1 = -1.000000e+00
x0 = 1.000000e+00

```



```

        int[] [] asub,
        int[] ptrb,
        int[] ptre,
        int numvar,
        int[] basis
    )
{
    // Since the value infinity is never used, we define
    // 'infinity' symbolic purposes only
    double
    infinity = 0;

    mosek.stakey[] skx = new mosek.stakey [numvar];
    mosek.stakey[] skc = new mosek.stakey [numvar];

    for (int i = 0; i < numvar ; ++i)
    {
        skx[i] = mosek.stakey.bas;
        skc[i] = mosek.stakey.fix;
    }

    task.appendvars(numvar);
    task.appendcons(numvar);

    for (int i = 0; i < numvar ; ++i)
        task.putacol(i,
                      asub[i],
                      aval[i]);

    for (int i = 0 ; i < numvar ; ++i)
        task.putconbound(
            i,
            mosek.boundkey.fx,
            0.0,
            0.0);

    for (int i = 0 ; i < numvar ; ++i)
        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);
}

public static void Main ()
{

```

```

const int numcon = 2;
const int numvar = 2;

double[] []
aval   = new double[numvar] [];

aval[0] = new double[] { -1.0 };
aval[1] = new double[] { 1.0, 1.0};

int[] []
asub = new int[numvar] [];

asub[0] = new int[] {1};
asub[1] = new int[] {0, 1};

int []      ptrb  = {0, 1};
int []      ptre  = {1, 3};

int[]       bsub  = new int[numvar];
double[]    b     = new double[numvar];
int[]       basis = new int[numvar];

try
{
    using (mosek.Env env = new mosek.Env())
    {
        using (mosek.Task task = new mosek.Task(env))
        {
            // Directs the log task stream to the user specified
            // method task_msg_obj.streamCB
            task.set_Stream(mosek.streamtype.log, new msgclass ("[task]"));
            /* Put A matrix and factor A.
               Call this function only once for a given task. */

            setup(
                task,
                aval,
                asub,
                ptrb,
                ptre,
                numvar,
                basis
            );

            /* now solve rhs */
            b[0] = 1;
            b[1] = -2;
            bsub[0] = 0;
            bsub[1] = 1;
            int nz = 2;

            task.solvewithbasis(0, ref nz, bsub, b);
            Console.WriteLine ("\nSolution to Bx = b:\n\n");

            /* Print solution and show correspondents

```

(continues on next page)

```

        to original variables in the problem */
    for (int i = 0; i < nz; ++i)
    {
        if (basis[bsub[i]] < numcon)
            Console.WriteLine ("This should never happen\n");
        else
            Console.WriteLine ("x{0} = {1}\n", basis[bsub[i]] - numcon ,
↪b[bsub[i]] );
    }

    b[0] = 7;
    bsub[0] = 0;
    nz = 1;

    task.solvewithbasis(0, ref nz, bsub, b);

    Console.WriteLine ("\nSolution to Bx = b:\n\n");
    /* Print solution and show correspondents
       to original variables in the problem */
    for (int i = 0; i < nz; ++i)
    {
        if (basis[bsub[i]] < numcon)
            Console.WriteLine ("This should never happen\n");
        else
            Console.WriteLine ("x{0} = {1}\n", basis[bsub[i]] - numcon ,
↪b[bsub[i]] );
    }
    }
    }
    }
    catch (mosek.Exception e)
    {
        Console.WriteLine (e.Code);
        Console.WriteLine (e);
        throw;
    }
    }
}

```

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



```

        1.0, 1.0, 1.0,
        1.0, 1.0, 1.0
    };

    double[] C = {1.0, 2.0, 3.0,
                  4.0, 5.0, 6.0
    };

    double[] D = {1.0, 1.0, 1.0, 1.0};
    double[] Q = {1.0, 0.0, 0.0, 2.0};
    double[] v = new double[2];

    double xy;

    using (mosek.Env env = new mosek.Env())
    {
        /* BLAS routines */

        try
        {
            env.dot(n, x, y, out xy);

            env.axpy(n, alpha, x, y);

            env.gemv(mosek.transpose.no, m, n, alpha, A, x, beta, z);

            env.gemm(mosek.transpose.no, mosek.transpose.no, m, n, k, alpha, A, B, beta,
→ C);

            env.syrk(mosek.uplo.lo, mosek.transpose.no, m, k, alpha, A, beta, D);

            /* LAPACK routines*/

            env.potrf(mosek.uplo.lo, m, Q);

            env.syeig(mosek.uplo.lo, m, Q, v);

            env.syevd(mosek.uplo.lo, m, Q, v);
        }
        catch (mosek.Exception e)
        {
            Console.WriteLine (e.Code);
            Console.WriteLine (e);
        }
        finally
        {
            if (env != null) env.Dispose ();
        }
    }
}
}
}

```













# Chapter 10

## Technical guidelines

This section contains some more in-depth technical guidelines for Optimizer API for .NET, not strictly necessary for basic use of **MOSEK**.

### 10.1 Memory management and garbage collection

Users who experience memory leaks, especially:

- memory usage not decreasing after the solver terminates,
- memory usage increasing when solving a sequence of problems,

should make sure that the *Task* objects are properly garbage collected. Since each *Task* object links to a **MOSEK** task resource in a linked library, it is sometimes the case that the garbage collector is unable to reclaim it automatically. This means that substantial amounts of memory may be leaked. For this reason it is very important to make sure that the *Task* object is disposed of, either automatically or manually, when it is not used any more.

The *Task* class supports the *Context Manager* protocol, so it will be destroyed properly when used in a `using` statement:

```
using (env = new mosek.Env())
{
    // Create a task object.
    using (task = new mosek.Task(env, 0, 0))
    {
        // Construct the problem
        // ...
    }
}
```

If this is not possible, then the necessary cleanup is performed by the methods *Task.Dispose* and *Env.Dispose* which should be called explicitly.

### 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](#).





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*.
- 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**.

## 10.6 Deployment

When redistributing a .NET application using the **MOSEK** Optimizer API for .NET 9.3.21, the following libraries must be included:

64-bit Windows	32-bit Windows
mosek64_9_3.dll	mosek9_3.dll
mosekxx9_3.dll	mosekxx9_3.dll
cilkrts20.dll	cilkrts20.dll
mosekdotnet9_3.dll	mosekdotnet9_3.dll

# Chapter 11

## Case Studies

In this section we present some case studies in which the Optimizer API for .NET 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, 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 the **MOSEK** optimizer API.

- *Basic Markowitz model*
- *Efficient frontier*
- *Factor model and efficiency*
- *Market impact costs*
- *Transaction costs*
- *Cardinality constraints*









```

    {0.1667, 0.0232, 0.0013},
    {0.0000, 0.1033, -0.0022},
    {0.0000, 0.0000, 0.0338}
};
double[] x0 = {0.0, 0.0, 0.0};
double w = 1.0;
double totalBudget;

int numvar = 2 * n + 1;
int numcon = n + 1;

// Offset of variables into the API variable.
int offsetx = 0;
int offsets = n;
int offsett = n + 1;

// Make mosek environment.
using (mosek.Env env = new mosek.Env())
{
    // Create a task object.
    using (mosek.Task task = new mosek.Task(env, 0, 0))
    {
        // Directs the log task stream
        task.set_Stream(mosek.streamtype.log, new msgclass (""));

        // Constraints.
        task.appendcons(numcon);

        // Constraint bounds. Compute total budget.
        totalBudget = w;
        for (int i = 0; i < n; ++i)
        {
            totalBudget += x0[i];
            /* Constraint bounds  $c^l = c^u = 0$  */
            task.putconbound(i + 1, mosek.boundkey.fx, 0.0, 0.0);
            task.putconname(i + 1, "GT[" + (i + 1) + "]");
        }
        /* The total budget constraint  $c^l = c^u = \text{totalBudget}$  in first row of A. */
        task.putconbound(0, mosek.boundkey.fx, totalBudget, totalBudget);
        task.putconname(0, "budget");

        // Variables.
        task.appendvars(numvar);

        /* x variables. */
        for (int j = 0; j < n; ++j)
        {
            /* Return of asset j in the objective */
            task.putcj(offsetx + j, mu[j]);
            /* Coefficients in the first row of A */
            task.putaij(0, offsetx + j, 1.0);
            /* No short-selling -  $x^l = 0$ ,  $x^u = \text{inf}$  */
            task.putvarbound(offsetx + j, mosek.boundkey.lo, 0.0, infinity);
            task.putvarname(offsetx + j, "x[" + (j + 1) + "]");
        }
    }
}

```

```

/* s variable is a constant equal to gamma. */
task.putvarbound(offsets, mosek.boundkey.fx, gamma, gamma);
task.putvarname(offsets, "s");

/* t variables (t = GT*x). */
for (int j = 0; j < n; ++j)
{
    /* Copying the GT matrix in the appropriate block of A */
    for (int k = 0; k < n; ++k)
        if ( GT[k,j] != 0.0 )
            task.putaij(1 + k, offsetx + j, GT[k,j]);
    /* Diagonal -1 entries in a block of A */
    task.putaij(1 + j, offsett + j, -1.0);
    /* Free - no bounds */
    task.putvarbound(offsett + j, mosek.boundkey.fr, -infinity, infinity);
    task.putvarname(offsett + j, "t[" + (j + 1) + "]");
}

/* Define the cone spanned by (s, t), i.e. of dimension n + 1 */
int[] csub = new int[n + 1];
csub[0] = offsets;
for(int j = 0; j < n; j++) csub[j + 1] = offsett + j;
task.appendcone( mosek.conetype.quad,
                 0.0, /* For future use only, can be set to 0.0 */
                 csub );
task.putconename(0, "stddev");

/* A maximization problem */
task.putobjsense(mosek.objsense.maximize);

task.optimize();

/* Display solution summary for quick inspection of results */
task.solutionsummary(mosek.streamtype.log);

task.writedata("dump.opf");

/* Read the results */
double expret = 0.0;
double[] xx = new double[n + 1];

task.getxxslice(mosek.soltype.itr, 0, offsets + 1, xx);
for (int j = 0; j < n; ++j)
    expret += mu[j] * xx[j + offsetx];

Console.WriteLine("\nExpected return {0:E} for gamma {1:E}", expret,
    xx[offsets]);
}
}
}
}
}
}

```

The above code produces the result:



## Debugging Tips

Implementing an optimization model in Optimizer API for .NET can be error-prone. In order to check the code for accidental errors it is very useful to dump the problem to a file in a human readable form for visual inspection. The line

```
task.writedata("dump.opf");
```

does that and it produces a file with the content:

Listing 11.3: Problem (11.5) stored in OPF format.

```
[comment]
  Written by MOSEK version 8.1.0.24
  Date 11-09-17
  Time 14:34:24
[/comment]

[hints]
[hint NUMVAR] 7 [/hint]
[hint NUMCON] 4 [/hint]
[hint NUMANZ] 12 [/hint]
[hint NUMQNZ] 0 [/hint]
[hint NUMCONE] 1 [/hint]
[/hints]

[variables disallow_new_variables]
  'x[1]' 'x[2]' 'x[3]' s 't[1]'
  't[2]' 't[3]'
[/variables]

[objective maximize]
  1.073e-01 'x[1]' + 7.37e-02 'x[2]' + 6.270000000000001e-02 'x[3]'
[/objective]

[constraints]
[con 'budget'] 'x[1]' + 'x[2]' + 'x[3]' = 1e+00 [/con]
[con 'GT[1]'] 1.667e-01 'x[1]' + 2.32e-02 'x[2]' + 1.3e-03 'x[3]' - 't[1]' = 0e+00
→ [/con]
[con 'GT[2]'] 1.033e-01 'x[2]' - 2.2e-03 'x[3]' - 't[2]' = 0e+00 [/con]
[con 'GT[3]'] 3.38e-02 'x[3]' - 't[3]' = 0e+00 [/con]
[/constraints]

[bounds]
[b] 0e+00 <= 'x[1]','x[2]','x[3]' [/b]
[b] s = 5e-02 [/b]
[b] 't[1]','t[2]','t[3]' free [/b]
[cone quad 'stddev'] s, 't[1]', 't[2]', 't[3]' [/cone]
[/bounds]
```

Since the API variables have been given meaningful names it is easy to verify by hand that the model is correct.



```

    prefix = prfx;
}

public override void streamCB (string msg)
{
    Console.Write ("{0}{1}", prefix, msg);
}
}

public class portfolio_2_frontier
{
    public static void Main (String[] args)
    {
        const int n = 3;

        // Since the value infinity is never used, we define
        // 'infinity' symbolic purposes only
        double infinity = 0;
        double[] mu = {0.1073, 0.0737, 0.0627};
        double[,] GT = {
            {0.1667, 0.0232, 0.0013},
            {0.0000, 0.1033, -0.0022},
            {0.0000, 0.0000, 0.0338}
        };
        double[] x0 = {0.0, 0.0, 0.0};
        double w = 1.0;
        double[] alphas = {0.0, 0.25, 0.5, 0.75, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5}
↪;

        int numalphas = 12;

        int numvar = 2 * n + 2;
        int numcon = n + 1;

        //Offset of variables into the API variable.
        int offsetx = 0;
        int offsets = n;
        int offsett = n + 1;
        int offsetu = 2 * n + 1;

        // Make mosek environment.
        using (mosek.Env env = new mosek.Env())
        {
            // Create a task object.
            using (mosek.Task task = new mosek.Task(env, 0, 0))
            {
                // Directs the log task stream to the user specified
                // method msgclass.streamCB
                task.set_Stream (mosek.streamtype.log, new msgclass (""));

                task.appendvars(numvar);
                task.appendcons(numcon);

                //Constraints.
                for (int i = 1; i <= n; ++i)
                {
                    w += x0[i - 1];

```

(continues on next page)

```

    task.putconbound(i, mosek.boundkey.fx, 0.0, 0.0);
    task.putconname(i, "GT[" + i + "]");
}
for (int i = 0; i < n; ++i)
    task.putaij(0, offsetx + i, 1.0);
task.putconbound(0, mosek.boundkey.fx, w, w);
task.putconname(0, "budget");

// Objective
for(int i = 0; i < n; i++)
    task.putcj(offsetx + i, mu[i]);

for (int i = 0; i < n; ++i)
{
    // t = GT * x
    for (int j = i; j < n; ++j)
        task.putaij(i + 1, offsetx + j, GT[i, j]);

    // Budget constraint
    task.putaij(i + 1, offsett + i, -1.0);

    // x variable
    task.putvarbound(offsetx + i, mosek.boundkey.lo, 0.0, 0.0);

    task.putvarname(offsetx + i, "x[" + (i + 1) + "]");
    task.putvarname(offsett + i, "t[" + (i + 1) + "]");

    // t variable
    task.putvarbound(offsett + i, mosek.boundkey.fr, -infinity, infinity);
}

// s variable
task.putvarbound(offsets, mosek.boundkey.fr, -infinity, infinity);
task.putvarname(offsets, "s");

// u variable
task.putvarbound(offsetu, mosek.boundkey.fx, 0.5, 0.5);
task.putvarname(offsetu, "u");

//Cones.
int[] csub = new int[n+2];
csub[0] = offsets;
csub[1] = offsetu;
for(int i = 0; i < n; i++)
    csub[i+2] = offsett + i;

task.appendcone( mosek.conetype.rquad,
                 0.0, /* For future use only, can be set to 0.0 */
                 csub);
task.putconename(0, "variance");

/* A maximization problem */
task.putobjsense(mosek.objsense.maximize);

//task.writedata("dump.opf");

```









```

int offsetx = 0;
int offsets = offsetx + n;
int offsett = offsets + 1;
int offsetc = offsett + n;
int offsetz = offsetc + n;
int offsetf = offsetz + n;

int numvar = 5 * n + 1;

int offset_con_budget = 0;
int offset_con_gx_t = offset_con_budget + 1;
int offset_con_abs1 = offset_con_gx_t + n;
int offset_con_abs2 = offset_con_abs1 + n;

int numcon = 3 * n + 1;

// Make mosek environment.
using (mosek.Env env = new mosek.Env())
{
    // Create a task object.
    using (mosek.Task task = new mosek.Task(env, 0, 0))
    {
        // Directs the log task stream to the user specified
        // method msgclass.streamCB
        task.set_Stream(mosek.streamtype.log, new msgclass(""));

        //Set up constraint bounds, names and variable coefficients
        task.appendcons(numcon);
        for (int i = 0; i < n; ++i)
        {
            w += x0[i];
            task.putconbound(offset_con_gx_t + i, mosek.boundkey.fx, 0.0, 0.0);
            task.putconname(offset_con_gx_t + i, "GT[" + (i + 1) + "]");

            task.putconbound(offset_con_abs1 + i, mosek.boundkey.lo, -x0[i], infinity);
            task.putconname(offset_con_abs1 + i, "zabs1[" + (i + 1) + "]");

            task.putconbound(offset_con_abs2 + i, mosek.boundkey.lo, x0[i], infinity);
            task.putconname(offset_con_abs2 + i, "zabs2[" + (i + 1) + "]");
        }
        // e x = w + e x0
        task.putconbound(offset_con_budget, mosek.boundkey.fx, w, w);
        task.putconname(offset_con_budget, "budget");

        //Variables.
        task.appendvars(numvar);

        //the objective function coefficients
        for (int i = 0; i < n; i++)
        {
            task.putcj(offsetx + i, mu[i]);
        }

        double[] one_m_one = { 1.0, -1.0 };

```

(continues on next page)

```

double[] one_one = { 1.0, 1.0 };

//set up variable bounds and names
for (int i = 0; i < n; ++i)
{
    task.putvarbound(offsetx + i, mosek.boundkey.lo, 0.0, infinity);
    task.putvarbound(offsett + i, mosek.boundkey.fr, -infinity, infinity);
    task.putvarbound(offsetc + i, mosek.boundkey.fr, -infinity, infinity);
    task.putvarbound(offsetz + i, mosek.boundkey.fr, -infinity, infinity);
    task.putvarbound(offsetf + i, mosek.boundkey.fx, 1.0, 1.0);

    task.putvarname(offsetx + i, "x[" + (i + 1) + "]" );
    task.putvarname(offsett + i, "t[" + (i + 1) + "]" );
    task.putvarname(offsetc + i, "c[" + (i + 1) + "]" );
    task.putvarname(offsetz + i, "z[" + (i + 1) + "]" );
    task.putvarname(offsetf + i, "f[" + (i + 1) + "]" );

    for (int j = i; j < n; ++j)
        task.putaij(offset_con_gx_t + i, j, GT[i, j]);

    task.putaij(offset_con_gx_t + i, offsett + i, -1.0);

    task.putaij(offset_con_budget, offsetx + i, 1.0);
    task.putaij(offset_con_budget, offsetc + i, m[i]);

    // z_j - x_j >= -x0_j
    int[] indx1 = { offsetz + i, offsetx + i };
    task.putarow(offset_con_abs1 + i, indx1, one_m_one);
    // z_j + x_j >= +x0_j
    int[] indx2 = { offsetz + i, offsetx + i };
    task.putarow(offset_con_abs2 + i, indx2, one_one);
}

task.putvarbound(offsets, mosek.boundkey.fx, gamma, gamma);
task.putvarname(offsets, "s");

// Quaadratic cone
int[] csub = new int[n+1];
csub[0] = offsets;
for(int i = 0; i < n; i++) csub[i + 1] = offsett + i;
task.appendcone(mosek.conetype.quad, 0.0, csub);
task.putconename(0, "stddev");

// Power cones
for (int j = 0; j < n; ++j)
{
    int[] coneindx = { offsetc + j, offsetf + j, offsetz + j };
    task.appendcone(mosek.conetype.ppow, 2.0/3.0, coneindx);
    task.putconename(1 + j, "trans[" + (j + 1) + "]" );
}

/* A maximization problem */
task.putobjsense(mosek.objsense.maximize);

//Turn all log output off.
//task.putintparam(mosek.iparam.log,0);

```

```

//task.writedata("dump.opf");
/* Solve the problem */
task.optimize();

task.solutionsummary(mosek.streamtype.log);

double expret = 0.0;
double[] xx = new double[numvar];

task.getxx(mosek.soltype.itr, xx);

for (int j = 0; j < n; ++j)
    expret += mu[j] * xx[j + offsetx];

Console.WriteLine("Expected return {0:E6} for gamma {1:E6}\n\n", expret,
↪xx[offsets]);
    }
}
}
}
}
}

```

The example code above produces the result

```

Interior-point solution summary
Problem status : PRIMAL_AND_DUAL_FEASIBLE
Solution status : OPTIMAL
Primal.  obj: 7.4390639578e-02    nrm: 1e+00    Viol.  con: 1e-08    var: 0e+00    ↪
↪cones: 7e-09
Dual.    obj: 7.4390755614e-02    nrm: 3e-01    Viol.  con: 1e-19    var: 3e-08    ↪
↪cones: 0e+00

Expected return 7.439064e-02 for gamma 5.000000e-02

```

If the problem is dumped to an OPF file, it has the following content.

Listing 11.7: OPF file for problem (11.17).

```

[comment]
  Written by MOSEK version 9.0.0.31
  Date 10-01-18
  Time 12:10:24
[/comment]

[hints]
[hint NUMVAR] 16 [/hint]
[hint NUMCON] 10 [/hint]
[hint NUMANZ] 27 [/hint]
[hint NUMQNZ] 0 [/hint]
[hint NUMCONE] 4 [/hint]
[/hints]

[variables disallow_new_variables]
'x[1]' 'x[2]' 'x[3]' 's' 't[1]'
't[2]' 't[3]' 'c[1]' 'c[2]' 'c[3]'
'z[1]' 'z[2]' 'z[3]' 'f[1]' 'f[2]'
'f[3]'

```

(continues on next page)





```

{
    string prefix;
    public msgclass (string prfx)
    {
        prefix = prfx;
    }

    public override void streamCB (string msg)
    {
        Console.Write ("{0}{1}", prefix, msg);
    }
}

public class portfolio_4_transcost
{
    public static void Main (String[] args)
    {
        const int n = 3;

        // Since the value infinity is never used, we define
        // 'infinity' symbolic purposes only
        double infinity = 0;
        double gamma = 0.05;
        double[] mu = {0.1073, 0.0737, 0.0627};
        double[,] GT = {
            {0.1667, 0.0232, 0.0013},
            {0.0000, 0.1033, -0.0022},
            {0.0000, 0.0000, 0.0338}
        };
        double[] x0 = {0.0, 0.0, 0.0};
        double w = 1.0;
        double[] f = {0.01, 0.01, 0.01};
        double[] g = {0.001, 0.001, 0.001};

        // Total initial wealth
        double U = w;
        for(int i=0; i< n; i++) U += x0[i];

        int offsetx = 0;
        int offsets = offsetx + n;
        int offsett = offsets + 1;
        int offsetz = offsett + n;
        int offsety = offsetz + n;

        int numvar = 4 * n + 1;

        int offset_con_budget = 0;
        int offset_con_gx_t = offset_con_budget + 1;
        int offset_con_abs1 = offset_con_gx_t + n;
        int offset_con_abs2 = offset_con_abs1 + n;
        int offset_con_ind = offset_con_abs2 + n;

        int numcon = 4 * n + 1;

        // Make mosek environment.
        using (mosek.Env env = new mosek.Env())

```

(continues on next page)

```

{
    // Create a task object.
    using (mosek.Task task = new mosek.Task(env, 0, 0))
    {
        // Directs the log task stream to the user specified
        // method msgclass.streamCB
        task.set_Stream(mosek.streamtype.log, new msgclass(""));

        //Set up constraint bounds, names and variable coefficients
        task.appendcons(numcon);
        for (int i = 0; i < n; ++i)
        {
            task.putconbound(offset_con_gx_t + i, mosek.boundkey.fx, 0.0, 0.0);
            task.putconname(offset_con_gx_t + i, "GT[" + (i + 1) + "]");

            task.putconbound(offset_con_abs1 + i, mosek.boundkey.lo, -x0[i], infinity);
            task.putconname(offset_con_abs1 + i, "zabs1[" + (i + 1) + "]");

            task.putconbound(offset_con_abs2 + i, mosek.boundkey.lo, x0[i], infinity);
            task.putconname(offset_con_abs2 + i, "zabs2[" + (i + 1) + "]");

            task.putconbound(offset_con_ind + i, mosek.boundkey.up, -infinity, 0.0);
            task.putconname(offset_con_ind + i, "ind[" + (i + 1) + "]");
        }
        //  $e x = w + e x_0$ 
        task.putconbound(offset_con_budget, mosek.boundkey.fx, U, U);
        task.putconname(offset_con_budget, "budget");

        //Variables.
        task.appendvars(numvar);

        //the objective function coefficients
        for (int i = 0; i < n; i++)
        {
            task.putcj(offsetx + i, mu[i]);
        }

        double[] one_m_one = { 1.0, -1.0 };
        double[] one_one = { 1.0, 1.0 };

        //set up variable bounds and names
        for (int i = 0; i < n; ++i)
        {
            task.putvarbound(offsetx + i, mosek.boundkey.lo, 0.0, infinity);
            task.putvarbound(offsett + i, mosek.boundkey.fr, -infinity, infinity);
            task.putvarbound(offsetz + i, mosek.boundkey.fr, -infinity, infinity);
            task.putvarbound(offsety + i, mosek.boundkey.ra, 0.0, 1.0);
            task.putvartype(offsety + i, mosek.variabletype.type_int);

            task.putvarname(offsetx + i, "x[" + (i + 1) + "]");
            task.putvarname(offsett + i, "t[" + (i + 1) + "]");
            task.putvarname(offsetz + i, "z[" + (i + 1) + "]");
            task.putvarname(offsety + i, "y[" + (i + 1) + "]");

            for (int j = i; j < n; ++j)

```

(continues on next page)

```

        task.putaij(offset_con_gx_t + i, j, GT[i, j]);

        task.putaij(offset_con_gx_t + i, offsett + i, -1.0);

        task.putaij(offset_con_budget, offsetx + i, 1.0);
        task.putaij(offset_con_budget, offsetz + i, g[i]);
        task.putaij(offset_con_budget, offsety + i, f[i]);

        // z_j - x_j >= -x0_j
        int[] indx1 = { offsetz + i, offsetx + i };
        task.putarow(offset_con_abs1 + i, indx1, one_m_one);
        // z_j + x_j >= +x0_j
        int[] indx2 = { offsetz + i, offsetx + i };
        task.putarow(offset_con_abs2 + i, indx2, one_one);
        // z_j - U*y_j <= 0
        int[] indx3 = { offsetz + i, offsety + i };
        double[] va = { 1.0, -U };
        task.putarow(offset_con_ind + i, indx3, va);
    }

    task.putvarbound(offsets, mosek.boundkey.fx, gamma, gamma);
    task.putvarname(offsets, "s");

    // Quaadratic cone
    int[] csub = new int[n+1];
    csub[0] = offsets;
    for(int i = 0; i < n; i++) csub[i + 1] = offsett + i;
    task.appendcone(mosek.conetype.quad, 0.0, csub);
    task.putconename(0, "stddev");

    /* A maximization problem */
    task.putobjsense(mosek.objsense.maximize);

    //Turn all log output off.
    //task.putintparam(mosek.iparam.log, 0);

    //task.writedata("dump.opf");
    /* Solve the problem */
    task.optimize();

    task.solutionsummary(mosek.streamtype.log);

    double expret = 0.0;
    double[] xx = new double[numvar];

    task.getxx(mosek.soltype.itg, xx);

    for (int j = 0; j < n; ++j)
        expret += mu[j] * xx[j + offsetx];

    Console.WriteLine("Expected return {0:E6} for gamma {1:E6}\n\n", expret,
        xx[offsets]);
    }
    }
    }
}

```



## Example code

The following example code demonstrates how to compute an optimal portfolio with cardinality bounds. Note that we are now solving a problem with integer variables, and therefore the solution must be retrieved from *soltype.itg*.

Listing 11.9: Code solving problem (11.20).

```
public static double[] markowitz_with_card(int n,
                                           double[] x0,
                                           double w,
                                           double gamma,
                                           double[] mu,
                                           double[,] GT,
                                           int k)
{
    // Total initial wealth
    double U = w;
    for(int i=0; i< n; i++) U += x0[i];

    int offsetx = 0;
    int offsets = offsetx + n;
    int offsett = offsets + 1;
    int offsetz = offsett + n;
    int offsety = offsetz + n;

    int numvar = 4 * n + 1;

    int offset_con_budget = 0;
    int offset_con_gx_t = offset_con_budget + 1;
    int offset_con_abs1 = offset_con_gx_t + n;
    int offset_con_abs2 = offset_con_abs1 + n;
    int offset_con_ind = offset_con_abs2 + n;
    int offset_con_card = offset_con_ind + n;

    int numcon = 4 * n + 2;

    // Make mosek environment.
    using (mosek.Env env = new mosek.Env())
    {
        // Create a task object.
        using (mosek.Task task = new mosek.Task(env, 0, 0))
        {
            // Directs the log task stream to the user specified
            // method msgclass.streamCB
            task.set_Stream(mosek.streamtype.log, new msgclass(""));

            //Set up constraint bounds, names and variable coefficients
            task.appendcons(numcon);
            for (int i = 0; i < n; ++i)
            {
                task.putconbound(offset_con_gx_t + i, mosek.boundkey.fx, 0.0, 0.0);
                task.putconname(offset_con_gx_t + i, "GT[" + (i + 1) + "]");

                task.putconbound(offset_con_abs1 + i, mosek.boundkey.lo, -x0[i],
→infinity);
                task.putconname(offset_con_abs1 + i, "zabs1[" + (i + 1) + "]");
            }
        }
    }
}
```

(continues on next page)

```

task.putconbound(offset_con_abs2 + i, mosek.boundkey.lo, x0[i], infinity);
task.putconname(offset_con_abs2 + i, "zabs2[" + (i + 1) + "]");

task.putconbound(offset_con_ind + i, mosek.boundkey.up, -infinity, 0.0);
task.putconname(offset_con_ind + i, "ind[" + (i + 1) + "]");
}
// e x = w + e x0
task.putconbound(offset_con_budget, mosek.boundkey.fx, U, U);
task.putconname(offset_con_budget, "budget");

// sum(y) <= k
task.putconbound(offset_con_card, mosek.boundkey.up, -infinity, k);
task.putconname(offset_con_card, "cardinality");

//Variables.
task.appendvars(numvar);

//the objective function coefficients
for (int i = 0; i < n; i++)
{
    task.putcj(offsetx + i, mu[i]);
}

double[] one_m_one = { 1.0, -1.0 };
double[] one_one = { 1.0, 1.0 };

//set up variable bounds and names
for (int i = 0; i < n; ++i)
{
    task.putvarbound(offsetx + i, mosek.boundkey.lo, 0.0, infinity);
    task.putvarbound(offsett + i, mosek.boundkey.fr, -infinity, infinity);
    task.putvarbound(offsetz + i, mosek.boundkey.fr, -infinity, infinity);
    task.putvarbound(offsety + i, mosek.boundkey.ra, 0.0, 1.0);
    task.putvartype(offsety + i, mosek.variabletype.type_int);

    task.putvarname(offsetx + i, "x[" + (i + 1) + "]");
    task.putvarname(offsett + i, "t[" + (i + 1) + "]");
    task.putvarname(offsetz + i, "z[" + (i + 1) + "]");
    task.putvarname(offsety + i, "y[" + (i + 1) + "]");

    for (int j = i; j < n; ++j)
        task.putaij(offset_con_gx_t + i, j, GT[i, j]);

    task.putaij(offset_con_gx_t + i, offsett + i, -1.0);
    task.putaij(offset_con_budget, offsetx + i, 1.0);

    // z_j - x_j >= -x0_j
    int[] indx1 = { offsetz + i, offsetx + i };
    task.putarow(offset_con_abs1 + i, indx1, one_m_one);
    // z_j + x_j >= +x0_j
    int[] indx2 = { offsetz + i, offsetx + i };
    task.putarow(offset_con_abs2 + i, indx2, one_one);
    // z_j - U*y_j <= 0
    int[] indx3 = { offsetz + i, offsety + i };
    double[] va = { 1.0, -U };
    task.putarow(offset_con_ind + i, indx3, va);
}

```

```

        // sum(y)
        task.putaij(offset_con_card, offsety + i, 1.0);
    }

    task.putvarbound(offsets, mosek.boundkey.fx, gamma, gamma);
    task.putvarname(offsets, "s");

    // Quaadratic cone
    int[] csub = new int[n+1];
    csub[0] = offsets;
    for(int i = 0; i < n; i++) csub[i + 1] = offsett + i;
    task.appendcone(mosek.conetype.quad, 0.0, csub);
    task.putconename(0, "stddev");

    /* A maximization problem */
    task.putobjsense(mosek.objsense.maximize);

    //Turn all log output off.
    task.putintparam(mosek.iparam.log,0);

    //task.writedata("dump.opf");
    /* Solve the problem */
    task.optimize();
    task.solutionsummary(mosek.streamtype.log);

    double[] xx = new double[n];
    task.getxxslice(mosek.soltype.itg, offsetx, offsetx + n, xx);
    return xx;
}
}
}

```

If we solve our running example with  $k = 1, 2, 3$  then we get the following solutions, with increasing expected returns:

Bound 1:	x = 0.00000 0.00000 1.00000	Return:	x = 0.06270
Bound 2:	x = 0.25286 0.00000 0.74714	Return:	x = 0.07398
Bound 3:	x = 0.23639 0.13850 0.62511	Return:	x = 0.07477

## 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*.





```

public static double[] logisticRegression(Env    env,
                                         double[, ] X,
                                         bool[]  y,
                                         double  lamb)
{
    int n = X.GetLength(0);
    int d = X.GetLength(1);      // num samples, dimension

    using (Task task = new Task(env, 0, 0))
    {
        // Variables [r; theta; t; u]
        int nvar = 1+d+2*n;
        task.appendvars(nvar);
        task.putvarboundsliceconst(0, nvar, boundkey.fr, -inf, inf);
        int r = 0, theta = 1, t = 1+d, u = 1+d+n;

        // Constraints: theta'*X +/- u = 0
        task.appendcons(n);
        task.putconboundsliceconst(0, n, boundkey.fx, 0, 0);

        // Objective lambda*r + sum(t)
        task.putcj(r, lamb);
        for(int i = 0; i < n; i++)
            task.putcj(t+i, 1.0);

        // The X block in theta'*X +/- u = 0
        int[] subi = new int[d*n+n];
        int[] subj = new int[d*n+n];
        double[] aval = new double[d*n+n];
        int k = 0;
        for(int i = 0; i < n; i++)
        {
            for(int j = 0; j < d; j++)
            {
                subi[k] = i; subj[k] = theta+j; aval[k] = X[i,j]; k++;
            }
            subi[d*n+i] = i; subj[d*n+i] = u+i;
            if (y[i]) aval[d*n+i] = 1; else aval[d*n+i] = -1;
        }
        task.putaijlist(subi, subj, aval);

        // Softplus function constraints
        softplus(task, t, u, n);

        // Regularization
        task.appendconeseq(conetype.quad, 0.0, 1+d, r);

        // Solution
        task.optimize();
        double[] xx = new double[d];
        task.getxxslice(soltype.itr, theta, theta+d, xx);

        return xx;
    }
}

```



(continued from previous page)

```
public CallbackProxy()
{
    stop = false;
}

public override int progressCB(mosek.callbackcode caller)
{
    // Return non-zero implies terminate the optimizer
    return stop ? 1 : 0;
}

public bool Stop
{
    get { return stop; }
    set { stop = value; }
}
}
```

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 11.13: A routine for parallel task race.

```
public static bool optimize(mosek.Task[] tasks,
                           mosek.rescode[] res,
                           mosek.rescode[] trm,
                           out int firstOK)
{
    var n = tasks.Length;
    var jobs = new System.Threading.Tasks.Task[n];
    int firstStop = -1;

    // Set a callback function
    var cb = new CallbackProxy();
    for (var i = 0; i < n; ++i)
        tasks[i].set_Progress(cb);

    // Initialize
    for (var i = 0; i < n; ++i)
    {
        res[i] = mosek.rescode.err_unknown;
        trm[i] = mosek.rescode.err_unknown;
    }

    // Start parallel optimizations, one per task
    for (var i = 0; i < n; ++i)
    {
        int num = i;
        jobs[i] = System.Threading.Tasks.Task.Factory.StartNew( () => {
            try
            {
                trm[num] = tasks[num].optimize();
                res[num] = mosek.rescode.ok;
            }
            catch (mosek.Exception e)
            {
                trm[num] = mosek.rescode.err_unknown;
            }
        });
    }
}
```

(continues on next page)

```

        res[num] = e.Code;
    }
    finally
    {
        // If this finished with success, inform other tasks to interrupt
        if (res[num] == mosek.rescode.ok)
        {
            if (!cb.Stop) firstStop = num;
            cb.Stop = true;
        }
    }
} );
}

// Join all threads
foreach (var j in jobs)
    j.Wait();

// For debugging, print res and trm codes for all optimizers
for (var i = 0; i < n; ++i)
    Console.WriteLine("Optimizer {0} res {1} trm {2}", i, res[i], trm[i]);

firstOK = firstStop;
return cb.Stop;
}

```

### 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 11.14: Concurrent optimization with different optimizers.

```

public static int optimizeconcurrent(mosek.Task task,
                                     int[] optimizers,
                                     out mosek.Task winTask,
                                     out mosek.rescode winTrm,
                                     out mosek.rescode winRes)
{
    var n = optimizers.Length;
    var tasks = new mosek.Task[n];
    var res = new mosek.rescode[n];
    var trm = new mosek.rescode[n];

    // Clone tasks and choose various optimizers
    for (var i = 0; i < n; ++i)
    {
        tasks[i] = new mosek.Task(task);
        tasks[i].putintparam(mosek.iparam.optimizer, optimizers[i]);
    }

    // Solve tasks in parallel
    bool success;
    int firstOK;
    success = optimize(tasks, res, trm, out firstOK);
}

```

(continues on next page)

```

if (success)
{
    winTask = tasks[firstOK];
    winTrm  = trm[firstOK];
    winRes  = res[firstOK];
    return firstOK;
}
else
{
    winTask = null;
    winTrm  = 0;
    winRes  = 0;
    return -1;
}
}

```

It remains to call the method with a choice of optimizers, for example:

Listing 11.15: Calling concurrent linear optimization.

```

int[] optimizers = {
    mosek.optimizertype.conic,
    mosek.optimizertype.dual_simplex,
    mosek.optimizertype.primal_simplex
};

idx = optimizeconcurrent(task, optimizers, out t, out trm, out res);

```

### 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 matter which one terminated first. Instead we examine all the task clones for the best objective value.

Listing 11.16: Concurrent optimization of a mixed-integer problem.

```

public static int optimizeconcurrentMIO(mosek.Task task,
                                         int[] seeds,
                                         out mosek.Task winTask,
                                         out mosek.rescode winTrm,
                                         out mosek.rescode winRes)
{
    var n = seeds.Length;
    var tasks = new mosek.Task[n];
    var res = new mosek.rescode[n];
    var trm = new mosek.rescode[n];

    // Clone tasks and choose various seeds for the optimizer
    for (var i = 0; i < n; ++i)
    {
        tasks[i] = new mosek.Task(task);
        tasks[i].putintparam(mosek.iparam.mio_seed, seeds[i]);
    }
}

```

(continues on next page)

```

// Solve tasks in parallel
bool success;
int firstOK;
success = optimize(tasks, res, trm, out firstOK);

if (success)
{
    // Pick the task that ended with res = ok
    // and contains an integer solution with best objective value
    mosek.objsense sense = task.getobjsense();
    double bestObj = (sense == mosek.objsense.minimize) ? 1.0e+10 : -1.0e+10;
    int bestPos = -1;

    for (var i = 0; i < n; ++i)
        Console.WriteLine("{0} {1} ", i, tasks[i].getprimalobj(mosek.soltype.
→itg));

    for (var i = 0; i < n; ++i)
        if ((res[i] == mosek.rescode.ok) &&
            (tasks[i].getsolsta(mosek.soltype.itg) == mosek.solsta.prim_feas ||
            tasks[i].getsolsta(mosek.soltype.itg) == mosek.solsta.integer_optimal)
→&&
            ((sense == mosek.objsense.minimize) ?
                (tasks[i].getprimalobj(mosek.soltype.itg) < bestObj) :
                (tasks[i].getprimalobj(mosek.soltype.itg) > bestObj) ) )
        {
            bestObj = tasks[i].getprimalobj(mosek.soltype.itg);
            bestPos = i;
        }

    if (bestPos != -1)
    {
        winTask = tasks[bestPos];
        winTrm = trm[bestPos];
        winRes = res[bestPos];
        return bestPos;
    }
}

winTask = null;
winTrm = 0;
winRes = 0;
return -1;
}

```

It remains to call the method with a choice of seeds, for example:

Listing 11.17: Calling concurrent integer optimization.

```

int[] seeds = { 42, 13, 71749373 };

idx = optimizeconcurrentMIO(task, seeds, out t, out trm, out res);

```

# Chapter 12

## Problem Formulation and Solutions

In this chapter we will discuss the following issues:

- 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

$$\begin{array}{llllll} \text{minimize} & & c^T x + c^f & & & \\ \text{subject to} & l^c & \leq & Ax & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x, \end{array} \quad (12.1)$$

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*.

















- 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 (12.22) is given by

$$\begin{aligned}
& \text{maximize} && (l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x + \frac{1}{2} x^T \left\{ \sum_{k=0}^{m-1} y_k Q^k - Q^o \right\} x + c^f \\
& \text{subject to} && A^T y + s_l^x - s_u^x + \left\{ \sum_{k=0}^{m-1} y_k Q^k - Q^o \right\} x = c, \\
& && -y + s_l^c - s_u^c = 0, \\
& && s_l^c, s_u^c, s_l^x, s_u^x \geq 0.
\end{aligned} \tag{12.23}$$

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 (12.22) and the dual problem (12.23).

### 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.

# 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`. The two most time-consuming steps of the presolve are



## 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.

































## Caveats

Observe if the infeasible problem

$$\begin{array}{lll} \text{minimize} & x + z \\ \text{subject to} & x = -1, \\ & x \geq 0 \end{array}$$

is repaired then it will become unbounded. Hence, a repaired problem may not have an optimal solution.

Another and more important caveat is that only a minimal repair is performed i.e. the repair that barely makes the problem feasible. Hence, the repaired problem is barely feasible and that sometimes makes the repaired problem hard to solve.

## Using the automatic repair tool

In this subsection we consider an infeasible linear optimization example:

$$\begin{array}{llll} \text{minimize} & -10x_1 & -9x_2, \\ \text{subject to} & 7/10x_1 + 1x_2 \leq 630, \\ & 1/2x_1 + 5/6x_2 \leq 600, \\ & 1x_1 + 2/3x_2 \leq 708, \\ & 1/10x_1 + 1/4x_2 \leq 135, \\ & x_1, & x_2 \geq 0, \\ & & x_2 \geq 650. \end{array} \quad (14.2)$$

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).

```
using System;

namespace mosek.example
{
    class msgclass : mosek.Stream
    {
        public msgclass () {}

        public override void streamCB (string msg)
        {
            Console.Write ("{1}", msg);
        }
    }

    public class feasrepairx1
    {
        public static void Main (String[] args)
        {
            string filename = "../data/feasrepair.lp";
            if (args.Length >= 1) filename = args[0];

            using (mosek.Env env = new mosek.Env())
            {
                using (mosek.Task task = new mosek.Task(env, 0, 0))
                {
                    task.set_Stream (mosek.streamtype.log, new msgclass());

                    task.readdata(filename);
                }
            }
        }
    }
}
```

(continues on next page)

(continued from previous page)

```
task.putintparam(mosek.iparam.log_feas_repair, 3);

task.primalrepair(null, null, null, null);

double sum_viol = task.getdouinf(mosek.dinfitem.primal_repair_penalty_obj);

Console.WriteLine("Minimized sum of violations = %{0}", sum_viol);

task.optimize();
task.solutionsummary(mosek.streamtype.msg);
    }
}
}
}
```

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
Lin. dep. - tries	: 1	time	: 0.00
Lin. dep. - number	: 0		

Presolve terminated. Time: 0.00

(continues on next page)













```

{
class msgclass : mosek.Stream
{
    string prefix;
    public msgclass (string prfx)
    {
        prefix = prfx;
    }

    public override void streamCB (string msg)
    {
        Console.Write ("{0}{1}", prefix, msg);
    }
}

public class sensitivity
{
    public static void Main ()
    {
        const double
        infinity = 0;

        mosek.boundkey[] bkc = new mosek.boundkey[] {
            mosek.boundkey.up, mosek.boundkey.up,
            mosek.boundkey.up, mosek.boundkey.fx,
            mosek.boundkey.fx, mosek.boundkey.fx,
            mosek.boundkey.fx
        };

        mosek.boundkey[] bkc = new mosek.boundkey[] {
            mosek.boundkey.lo, mosek.boundkey.lo,
            mosek.boundkey.lo, mosek.boundkey.lo,
            mosek.boundkey.lo, mosek.boundkey.lo,
            mosek.boundkey.lo
        };

        int[] ptrb = new int[] {0, 2, 4, 6, 8, 10, 12};
        int[] ptre = new int[] {2, 4, 6, 8, 10, 12, 14};
        int[] sub = new int[] {0, 3, 0, 4, 1, 5, 1, 6, 2, 3, 2, 5, 2, 6};
        double[] blc = new double[] {
            -infinity, -infinity,
            -infinity, 800, 100, 500, 500
        };

        double[] buc = new double[] {400, 1200, 1000, 800, 100, 500, 500};
        double[] c = new double[] {1.0, 2.0, 5.0, 2.0, 1.0, 2.0, 1.0};
        double[] blx = new double[] {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
        double[] bux = new double[] {infinity,
            infinity,
            infinity,
            infinity,
            infinity,
            infinity,
            infinity
        };
    }
}

```

(continues on next page)

```

double[] val = new double[] {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
                             };

int numcon = 7;  /* Number of constraints.          */
int numvar = 7;  /* Number of variables.           */
try
{
    using (mosek.Env env = new mosek.Env())
    {
        using (mosek.Task task = new mosek.Task(env))
        {
            // Directs the log task stream to the user specified
            // method task_msg_obj.streamCB
            task.set_Stream(mosek.streamtype.log, new msgclass ("[task]"));

            task.inputdata(numcon, numvar,
                           c,
                           0.0,
                           ptrb,
                           ptre,
                           sub,
                           val,
                           bkc,
                           blc,
                           buc,
                           bkc,
                           blx,
                           blx,
                           bux);

            /* A maximization problem */
            task.putobjsense(mosek.objsense.minimize);

            try
            {
                task.optimize();
            }
            catch (mosek.Warning w)
            {
                Console.WriteLine("Mosek warning:");
                Console.WriteLine (w.Code);
                Console.WriteLine (w);
            }

            /* Analyze upper bound on c1 and the equality constraint on c4 */
            int[] subi = new int [] {0, 3};
            mosek.mark[] marki = new mosek.mark[] {mosek.mark.up,
                                                    mosek.mark.up
                                                    };

            /* Analyze lower bound on the variables x12 and x31 */
            int[] subj = new int [] {1, 4};
            mosek.mark[] markj = new mosek.mark[] {mosek.mark.lo,
                                                    mosek.mark.lo
                                                    };

```

```

double[] leftpricei = new double[2];
double[] rightpricei = new double[2];
double[] leftrangei = new double[2];
double[] rightrangei = new double[2];
double[] leftpricej = new double[2];
double[] rightpricej = new double[2];
double[] leftrangej = new double[2];
double[] rightrangej = new double[2];

task.primalsensitivity( subi,
                        marki,
                        subj,
                        markj,
                        leftpricei,
                        rightpricei,
                        leftrangei,
                        rightrangei,
                        leftpricej,
                        rightpricej,
                        leftrangej,
                        rightrangej);

Console.WriteLine("Results from sensitivity analysis on bounds:\n");

Console.WriteLine("For constraints:\n");
for (int i = 0; i < 2; ++i)
    Console.Write(
        "leftprice = {0}, rightprice = {1},leftrange = {2}, rightrange ={3}\n
↪",
        leftpricei[i], rightpricei[i], leftrangei[i], rightrangei[i]);

Console.WriteLine("For variables:\n");
for (int i = 0; i < 2; ++i)
    Console.Write(
        "leftprice = {0}, rightprice = {1},leftrange = {2}, rightrange ={3}\n
↪",
        leftpricej[i], rightpricej[i], leftrangej[i], rightrangej[i]);

double[] leftprice = new double[2];
double[] rightprice = new double[2];
double[] leftrange = new double[2];
double[] rightrange = new double[2];
int[] subc = new int[] {2, 5};

task.dualsensitivity( subc,
                    leftprice,
                    rightprice,
                    leftrange,
                    rightrange
                    );

Console.WriteLine("Results from sensitivity analysis on objective_
↪coefficients:");

```

(continues on next page)

```
        for (int i = 0; i < 2; ++i)
            Console.Write(
                "leftprice = {0}, rightprice = {1},leftrange = {2}, rightrange = {3}\n",
                leftprice[i], rightprice[i], leftrange[i], rightrange[i]);
        }
    }
}
catch (mosek.Exception e)
{
    Console.WriteLine (e.Code);
    Console.WriteLine (e);
    throw;
}
}
```

# Chapter 15

## API Reference

This section contains the complete reference of the **MOSEK** Optimizer API for .NET. 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 class types*
- *Nonlinear API (SCopt)*

### 15.1 API Conventions

#### 15.1.1 Function arguments

##### Naming Convention

In the definition of the **MOSEK** Optimizer API for .NET 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 .NET.

API name	API type	Dimension	Related problem parameter
numcon	int		$m$
numvar	int		$n$
numcone	int		$t$
ptrb	int[]	numvar	$a_{ij}$
ptre	int[]	numvar	$a_{ij}$
asub	int[]	ptre[numvar-1]	$a_{ij}$
aval	double[]	ptre[numvar-1]	$a_{ij}$
c	double[]	numvar	$c_j$
cfix	double		$c^f$
blc	double[]	numcon	$l_k^c$
buc	double[]	numcon	$u_k^c$
blx	double[]	numvar	$l_k^x$
bux	double[]	numvar	$u_k^x$
numqonz	int		$q_{ij}^o$
qosubi	int[]	numqonz	$q_{ij}^o$
qosubj	int[]	numqonz	$q_{ij}^o$
qoval	double[]	numqonz	$q_{ij}^o$
numqcnz	int		$q_{ij}^k$
qcsbk	int[]	numqcnz	$q_{ij}^k$
qcsubi	int[]	numqcnz	$q_{ij}^k$
qcsbj	int[]	numqcnz	$q_{ij}^k$
qcval	double[]	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:  $q_{qosubi[t],qosubj[t]}^o = qoval[t]$ ,  $t = 0, \dots, numqonz - 1$ .
- The linear terms in the objective :  $c_j = c[j]$ ,  $j = 0, \dots, numvar - 1$
- The fixed term in the objective :  $c^f = cfix$ .
- The quadratic terms in the constraints:  $q_{qcsbk[t],qcsbj[t]}^{qcsbk[t]} = qcval[t]$ ,  $t = 0, \dots, numqcnz - 1$
- The linear terms in the constraints:  $a_{asub[t],j} = aval[t]$ ,  $t = ptrb[j], \dots, ptre[j] - 1$ ,  $j = 0, \dots, numvar - 1$

### 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
<code>boundkey.fx</code>	finite	identical to the lower bound
<code>boundkey.fr</code>	minus infinity	plus infinity
<code>boundkey.lo</code>	finite	plus infinity
<code>boundkey.ra</code>	finite	finite
<code>boundkey.up</code>	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

$$l_k^c = \text{blc}[k], \quad k = 0, \dots, \text{numcon} - 1$$

$$u_k^c = \text{buc}[k], \quad k = 0, \dots, \text{numcon} - 1.$$

The bounds on the variables are specified using the variables `blkx`, `blx`, and `bux` in the same way. The numerical values for the lower bounds on the variables are given by

$$l_j^x = \text{blx}[j], \quad j = 0, \dots, \text{numvar} - 1.$$

$$u_j^x = \text{bux}[j], \quad j = 0, \dots, \text{numvar} - 1.$$

### 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

```
double[] c = new double[numvar];
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 constraints 2 through 9 (both inclusive) one would write

```
double[] upper_bound = new double[8];
double[] lower_bound = new double[8];
mosek.boundkey[] bound_key = new mosek.boundkey[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

```
int[] bound_index = { 1, 6, 3, 9 };
mosek.boundkey[] bound_key =
    { mosek.boundkey.fr,
      mosek.boundkey.lo,
      mosek.boundkey.up,
      mosek.boundkey.fx };
double[] lower_bound = { 0.0, -10.0, 0.0, 5.0 };
double[] 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:

```
int[] sub_i = { 1, 3, 5 };
int[] sub_j = { 2, 3, 4 };
double[] cof = { 1.1, 4.3, 0.2 };
task.putaijlist(sub_i, sub_j, 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:

$$a_{\text{asub}[k],j} = \text{aval}[k] \quad \text{for } j = 0, \dots, \text{numcol} - 1, k = \text{ptrb}[j], \dots, \text{ptre}[j] - 1.$$



## 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 object.
- *Infrequent: Task.set\_ItgSolutionCallback, Env.set\_Stream*

### Environment and task management

- *Env.Env* – Constructor of a new environment.
- *Task.Task* – Constructor of a new optimization task.
- *Task.puttaskname* – Assigns a new name to the task.
- *Infrequent: Task.Dispose, Env.Dispose, Task.commitchanges, Task.deletesolution, Task.putmaxnumanz, Task.putmaxnumbarvar, Task.putmaxnumcon, Task.putmaxnumcone, Task.putmaxnumqnz, Task.putmaxnumvar, Task.resizetask*

### Infeasibility diagnostic

- *Task.getinfeasiblesubproblem* – Obtains an infeasible subproblem.
- *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.
- *Infrequent: Task.getinfindex*

### Input/Output

- *Task.writedata* – Writes problem data to a file.
- *Task.writesolution* – Write a solution to a file.
- *Infrequent: Task.readdata, Task.readdataformat, Task.readjsonstring, Task.readlpstring, Task.readopfstring, Task.readparamfile, Task.readptfstring, Task.readsolution, Task.readsummary, Task.readtask, Task.writejsonsol, Task.writeparamfile, Task.writetask*



## Names

- *Env.getcodedesc* – Obtains a short description of a response code.
- *Task.putbarvarname* – Sets the name of a semidefinite variable.
- *Task.putconename* – Sets the name of a cone.
- *Task.putconname* – Sets the name of a constraint.
- *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.generateconenames*, *Task.generateconnames*, *Task.generatevarnames*, *Task.getbarvarname*, *Task.getbarvarnameindex*, *Task.getbarvarnamelen*, *Task.getconename*, *Task.getconenameindex*, *Task.getconenamelen*, *Task.getconname*, *Task.getconnameindex*, *Task.getconnamelen*, *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.strtoconetype*, *Task.strtosk*

## Optimization

- *Task.optimize* – Optimizes the problem.

## Parameters

- *Task.putdouparam* – Sets a double parameter.
- *Task.putintparam* – Sets an integer parameter.
- *Task.putparam* – Modifies the value of parameter.
- *Task.putstrparam* – Sets a string parameter.
- *Task.setdefaults* – Resets all parameter values.
- *Infrequent:* *Task.getatruncatetol*, *Task.getdouparam*, *Task.getintparam*, *Task.getnumparam*, *Task.getstrparam*, *Task.getstrparamlen*, *Task.isdouparname*, *Task.isintparname*, *Task.isstrparname*, *Task.putnadouparam*, *Task.putnaintparam*, *Task.putnastrparam*, *Task.readparamfile*, *Task.writeparamfile*

## 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

- *Task.appendcone* – Appends a new conic constraint to the problem.
- *Task.appendconeseq* – Appends multiple conic constraints to the problem.
- *Task.getnumcone* – Obtains the number of cones.
- *Task.putcone* – Replaces a conic constraint.
- *Task.putconename* – Sets the name of a cone.
- *Task.removecones* – Removes a number of conic constraints from the problem.
- *Infrequent:* *Task.appendconeseq*, *Task.generateconenames*, *Task.getcone*, *Task.getconeinfo*, *Task.getconename*, *Task.getconenameindex*, *Task.getconenamelen*, *Task.getmaxnumcone*, *Task.getnumconemem*, *Task.putmaxnumcone*

### 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 - 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.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.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.generateconnames*, *Task.generatevarnames*, *Task.getacol*, *Task.getacolnumnz*, *Task.getacolslice*, *Task.getacolslicenumnz*, *Task.getacolslicetrip*, *Task.getaij*, *Task.getapiecenunz*, *Task.getarow*, *Task.getarownumz*, *Task.getarowslice*, *Task.getarowslicenumz*, *Task.getarowslicetrip*, *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.getnumanz64*, *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.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.getnumqobjjnz*, *Task.getqconk*, *Task.getqobj*, *Task.getqobjij*, *Task.putmaxnumqnz*, *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.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.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.putbarablocktriplet*, *Task.putbarcblocktriplet*, *Task.putmaxnumanz*, *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.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 response from a remote job.
- *Task.asyncoptimize* – Offload the optimization task to a solver server.
- *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.
- *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.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.getreducedcosts*, *Task.getslc*, *Task.getslcslice*, *Task.getslx*, *Task.getslxslice*, *Task.getsnx*, *Task.getsnxslice*, *Task.getsolution*, *Task.getsolutionslice*, *Task.getsuc*, *Task.getsucslice*, *Task.getsux*, *Task.getsuxslice*, *Task.putconsolutioni*, *Task.putslc*, *Task.putslcslice*, *Task.putslx*, *Task.putslxslice*, *Task.putsnx*, *Task.putsnxslice*, *Task.putsolution*, *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.getsolution*, *Task.getsolutionslice*, *Task.getxc*, *Task.getxcslice*, *Task.putconsolutioni*, *Task.putsolution*, *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.onesolutionssummary* – Prints a short summary of a specified solution.
- *Task.solutiondef* – Checks whether a solution is defined.
- *Task.solutionssummary* – Prints a short summary of the current solutions.
- *Infrequent:* *Task.analyzesolution*, *Task.deletesolution*, *Task.getdualsolutionnorms*, *Task.getdviolbarvar*, *Task.getdviolcon*, *Task.getdviolcones*, *Task.getdviolvar*, *Task.getprimalsolutionnorms*, *Task.getpviolbarvar*, *Task.getpviolcones*, *Task.getskc*, *Task.getskcslice*, *Task.getskn*, *Task.getskx*, *Task.getskxslice*, *Task.getsolution*, *Task.getsolutionslice*, *Task.putconsolutioni*, *Task.putskc*, *Task.putskcslice*, *Task.putskx*, *Task.putskxslice*, *Task.putsolution*, *Task.putsolutionyi*, *Task.putvarsolutionj*

## Solving systems with basis matrix

- *Infrequent:* *Task.basiscond*, *Task.initbasissolve*, *Task.solvewithbasis*

## System, memory and debugging

- *Infrequent:* *Task.checkmem*, *Task.getmemusage*, *Env.setupthreads*

## Versions

- *Env.getversion* – Obtains MOSEK version information.
- *Infrequent:* *Env.getbuildinfo*

# 15.3 Class Env

`mosek.Env`

The **MOSEK** global environment.

`Env.Env`

Env()

Env(string dbgfile)

Constructor of a new environment.

**Parameters** `dbgfile` (`string`) – File where the memory debugging log is written. (input)

`Env.Dispose`

```
void Dispose ()
```

Free the underlying native allocation.

Env. `axpy`

```
void axpy
(int n,
 double alpha,
 double[] x,
 double[] y)
```

Adds  $\alpha x$  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** (int) – Length of the vectors. (input)
- **alpha** (double) – The scalar that multiplies  $x$ . (input)
- **x** (double[]) – The  $x$  vector. (input)
- **y** (double[]) – The  $y$  vector. (input/output)

**Groups** *Linear algebra*

Env. `checkinall`

```
void checkinall ()
```

Check in all unused license features to the license token server.

**Groups** *License system*

Env. `checkinlicense`

```
void checkinlicense (feature 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** (*feature*) – Feature to check in to the license system. (input)

**Groups** *License system*

Env. `checkoutlicense`

```
void checkoutlicense (feature 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* (*feature*) – Feature to check out from the license system. (input)

**Groups** *License system*

Env.computesparscholesky

```
void computesparscholesky
(int multithread,
 int ordermethod,
 double tolsingular,
 int[] anzc,
 long[] aptrc,
 int[] asubc,
 double[] avalc,
 out int[] perm,
 out double[] diag,
 out int[] lnzc,
 out long[] lptrc,
 out long lensubnval,
 out int[] lsubc,
 out double[] 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  $\rho$  is set equal to value of *tolsingular*.

#### Parameters

- **multithread** (int) – If nonzero then the function may exploit multiple threads. (input)
- **ordermethod** (int) – If nonzero, then a sparsity preserving ordering will be employed. (input)
- **tolsingular** (double) – A positive parameter controlling when a pivot is declared zero. (input)
- **anzc** (int[]) – **anzc**[j] is the number of nonzeros in the  $j$ -th column of  $A$ . (input)
- **aptrc** (long[]) – **aptrc**[j] is a pointer to the first element in column  $j$  of  $A$ . (input)
- **asubc** (int[]) – Row indexes for each column stored in increasing order. (input)

- `avalc (double[])` – The value corresponding to row indexed stored in `asubc`. (input)
- `perm (int[])` – Permutation array used to specify the permutation matrix  $P$  computed by the function. (output)
- `diag (double[])` – The diagonal elements of matrix  $D$ . (output)
- `lnzc (int[])` – `lnzc[j]` is the number of non zero elements in column  $j$  of  $L$ . (output)
- `lptrc (long[])` – `lptrc[j]` is a pointer to the first row index and value in column  $j$  of  $L$ . (output)
- `lensubnval (long)` – Number of elements in `lsubc` and `lvalc`. (output)
- `lsubc (int[])` – Row indexes for each column stored in increasing order. (output)
- `lvalc (double[])` – The values corresponding to row indexed stored in `lsubc`. (output)

**Groups** *Linear algebra*

`Env.dot`

```
void dot
(int n,
 double[] x,
 double[] y,
 out double 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 (int)` – Length of the vectors. (input)
- `x (double[])` – The  $x$  vector. (input)
- `y (double[])` – The  $y$  vector. (input)
- `xty (double)` – The result of the inner product between  $x$  and  $y$ . (output)

**Groups** *Linear algebra*

`Env.echointro`

```
void echointro (int longver)
```

Prints an intro to message stream.

**Parameters** `longver (int)` – If non-zero, then the intro is slightly longer. (input)

**Groups** *Logging*

`Env.gemm`

```
void gemm
(transpose transa,
 transpose transb,
 int m,
 int n,
 int k,
```

(continues on next page)

```
double alpha,
double[] a,
double[] b,
double beta,
double[] 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  $\alpha, \beta$  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** (`transpose`) – Indicates whether the matrix  $A$  must be transposed. (input)
- **transb** (`transpose`) – Indicates whether the matrix  $B$  must be transposed. (input)
- **m** (`int`) – Indicates the number of rows of matrix  $C$ . (input)
- **n** (`int`) – Indicates the number of columns of matrix  $C$ . (input)
- **k** (`int`) – 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** (`double`) – A scalar value multiplying the result of the matrix multiplication. (input)
- **a** (`double[]`) – The pointer to the array storing matrix  $A$  in a column-major format. (input)
- **b** (`double[]`) – The pointer to the array storing matrix  $B$  in a column-major format. (input)
- **beta** (`double`) – A scalar value that multiplies  $C$ . (input)
- **c** (`double[]`) – The pointer to the array storing matrix  $C$  in a column-major format. (input/output)

**Groups** *Linear algebra*

`Env.gemv`

```
void gemv
(transpose transa,
 int m,
 int n,
 double alpha,
 double[] a,
 double[] x,
 double beta,
 double[] 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  $\alpha, \beta$  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** (*transpose*) – Indicates whether the matrix  $A$  must be transposed. (input)
- **m** (int) – Specifies the number of rows of the matrix  $A$ . (input)
- **n** (int) – Specifies the number of columns of the matrix  $A$ . (input)
- **alpha** (double) – A scalar value multiplying the matrix  $A$ . (input)
- **a** (double[]) – A pointer to the array storing matrix  $A$  in a column-major format. (input)
- **x** (double[]) – A pointer to the array storing the vector  $x$ . (input)
- **beta** (double) – A scalar value multiplying the vector  $y$ . (input)
- **y** (double[]) – A pointer to the array storing the vector  $y$ . (input/output)

**Groups** *Linear algebra*

`Env.getbuildinfo`

```
static void getbuildinfo
(StringBuilder buildstate,
StringBuilder builddate)
```

Obtains build information.

#### Parameters

- **buildstate** (StringBuilder) – State of binaries, i.e. a debug, release candidate or final release. (output)
- **builddate** (StringBuilder) – Date when the binaries were built. (output)

**Groups** *Versions*

`Env.getcodedesc`

```
static void getcodedesc
(rescode code,
StringBuilder symname,
StringBuilder str)
```

Obtains a short description of the meaning of the response code given by **code**.

#### Parameters

- **code** (*rescode*) – A valid **MOSEK** response code. (input)
- **symname** (StringBuilder) – Symbolic name corresponding to **code**. (output)
- **str** (StringBuilder) – Obtains a short description of a response code. (output)

**Groups** *Names, Responses, errors and warnings*

`Env.getversion`

```
static void getversion
(out int major,
out int minor,
out int revision)
```

Obtains **MOSEK** version information.

#### Parameters

- **major** (int) – Major version number. (output)

- `minor (int)` – Minor version number. (output)
- `revision (int)` – Revision number. (output)

**Groups** *Versions*

`Env.licensecleanup`

```
static void 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`

```
void linkfiletostream
(streamtype whichstream,
 string filename,
 int append)
```

Sends all output from the stream defined by `whichstream` to the file given by `filename`.

**Parameters**

- `whichstream (streamtype)` – Index of the stream. (input)
- `filename (string)` – A valid file name. (input)
- `append (int)` – If this argument is 0 the file will be overwritten, otherwise it will be appended to. (input)

**Groups** *Logging*

`Env.potrf`

```
void potrf
(uplo uplo,
 int n,
 double[] a)
```

Computes a Cholesky factorization of a real symmetric positive definite dense matrix.

**Parameters**

- `uplo (uplo)` – Indicates whether the upper or lower triangular part of the matrix is stored. (input)
- `n (int)` – Dimension of the symmetric matrix. (input)
- `a (double[])` – 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`

```
void putlicensecode (int[] code)
```

Input a runtime license code.

**Parameters** `code (int[])` – A runtime license code. (input)

**Groups** *License system*

Env.putlicensedebug

```
void putlicensedebug (int 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 (int)` – Whether license checkout debug info should be printed. (input)

**Groups** *License system*

Env.putlicensepath

```
void putlicensepath (string licensepath)
```

Set the path to the license file.

**Parameters** `licensepath (string)` – A path specifying where to search for the license. (input)

**Groups** *License system*

Env.putlicensawait

```
void putlicensawait (int 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 (int)` – Whether **MOSEK** should wait for a license if no license is available. (input)

**Groups** *License system*

Env.set\_Stream

```
void set_Stream  
(streamtype whichstream,  
Stream callback)
```

Directs all output from an environment stream to a callback object.

**Parameters**

- `whichstream (streamtype)` – Index of the stream. (input)
- `callback (Stream)` – The callback object. (input)

Env.setupthreads

```
void setupthreads (int numthreads)
```

Preallocates a thread pool for the interior-point and conic optimizers in the current process. This function should only be called once per process, before first optimization. Future settings of the parameter *iparam.num\_threads* will be irrelevant for the conic optimizer.

**Parameters** `numthreads (int)` – Number of threads. (input)

**Groups** *System, memory and debugging*

Env.sparsetriangularsolvedense

```

void sparsetriangarsolvedense
(transpose transposed,
 int[] lnzc,
 long[] lptrc,
 int[] lsubc,
 double[] lvalc,
 double[] 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

- **transpose** (*transpose*) – Controls whether to use with  $L$  or  $L^T$ . (input)
- **lnzc** (int[]) – **lnzc**[j] is the number of nonzeros in column j. (input)
- **lptrc** (long[]) – **lptrc**[j] is a pointer to the first row index and value in column j. (input)
- **lsubc** (int[]) – Row indexes for each column stored sequentially. Must be stored in increasing order for each column. (input)
- **lvalc** (double[]) – The value corresponding to the row index stored in **lsubc**. (input)
- **b** (double[]) – The right-hand side of linear equation system to be solved as a dense vector. (input/output)

**Groups** *Linear algebra*

Env.syeig

```

void syeig
(uplo uplo,
 int n,
 double[] a,
 double[] 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** (*uplo*) – Indicates whether the upper or lower triangular part is used. (input)
- **n** (int) – Dimension of the symmetric input matrix. (input)
- **a** (double[]) – A symmetric matrix  $A$  stored in column-major order. Only the part indicated by **uplo** is used. (input)
- **w** (double[]) – Array of length at least **n** containing the eigenvalues of  $A$ . (output)

**Groups** *Linear algebra*

Env.syevd

```
void syevd
(
    uplo uplo,
    int n,
    double[] a,
    double[] 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  $V = \text{diag}(w)$  and  $U$  contains the eigenvectors of  $A$ .

Note that the matrix  $U$  overwrites the input data  $A$ .

#### Parameters

- **uplo** (*uplo*) – Indicates whether the upper or lower triangular part is used. (input)
- **n** (int) – Dimension of the symmetric input matrix. (input)
- **a** (double[]) – 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** (double[]) – Array of length at least **n** containing the eigenvalues of  $A$ . (output)

**Groups** *Linear algebra*

Env.syrk

```
void syrk
(
    uplo uplo,
    transpose trans,
    int n,
    int k,
    double alpha,
    double[] a,
    double beta,
    double[] c)

```

Performs a symmetric rank- $k$  update for a symmetric matrix.

Given a symmetric matrix  $C \in \mathbb{R}^{n \times n}$ , two scalars  $\alpha, \beta$  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** (*uplo*) – Indicates whether the upper or lower triangular part of  $C$  is used. (input)

- **trans** (*transpose*) – Indicates whether the matrix  $A$  must be transposed. (input)
- **n** (**int**) – Specifies the order of  $C$ . (input)
- **k** (**int**) – Indicates the number of rows or columns of  $A$ , depending on whether or not it is transposed, and its rank. (input)
- **alpha** (**double**) – A scalar value multiplying the result of the matrix multiplication. (input)
- **a** (**double**[]) – The pointer to the array storing matrix  $A$  in a column-major format. (input)
- **beta** (**double**) – A scalar value that multiplies  $C$ . (input)
- **c** (**double**[]) – 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(Env env)
```

```
Task(
    Env env,
    int numcon,
    int numvar)
```

```
Task(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)

`Task.Dispose`

```
void Dispose()
```

Free the underlying native allocation.

`Task.analyzenames`

```
void analyzenames
(streamtype whichstream,
 nametype nametype)
```

The function analyzes the names and issues an error if a name is invalid.

### Parameters

- **whichstream** (*streamtype*) – Index of the stream. (input)

- **nametype** (*nametype*) – The type of names e.g. valid in MPS or LP files. (input)
- Groups** *Names*

**Task.**analyzeproblem

```
void analyzeproblem (streamtype whichstream)
```

The function analyzes the data of a task and writes out a report.

**Parameters** **whichstream** (*streamtype*) – Index of the stream. (input)

**Groups** *Inspecting the task*

**Task.**analyzesolution

```
void analyzesolution
(streamtype whichstream,
 soltype whichsol)
```

Print information related to the quality of the solution and other solution statistics.

By default this function prints information about the largest infeasibilities 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** (*streamtype*) – Index of the stream. (input)
- **whichsol** (*soltype*) – Selects a solution. (input)

**Groups** *Solution information, Inspecting the task*

**Task.**appendbarvars

```
void appendbarvars (int[] dim)
```

Appends positive semidefinite matrix variables of dimensions given by **dim** to the problem.

**Parameters** **dim** (int[]) – Dimensions of symmetric matrix variables to be added. (input)

**Groups** *Problem data - semidefinite*

**Task.**appendcone

```
void appendcone
(conetype ct,
 double coneapar,
 int[] submem)
```

```
void appendcone
(conetype ct,
 double coneapar,
 int nummem,
 int[] submem)
```





```

void appendsparsesymmat
(int dim,
 int[] subi,
 int[] subj,
 double[] valij,
 out long 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` (`int`) – Dimension of the symmetric matrix that is appended. (input)
- `subi` (`int[]`) – Row subscript in the triplets. (input)
- `subj` (`int[]`) – Column subscripts in the triplets. (input)
- `valij` (`double[]`) – Values of each triplet. (input)
- `idx` (`long`) – Unique index assigned to the inputted matrix that can be used for later reference. (output)

**Return** (`long`) – Unique index assigned to the inputted matrix that can be used for later reference.

**Groups** *Problem data - semidefinite*

`Task.appendsparsesymmatlist`

```

void appendsparsesymmatlist
(int[] dims,
 long[] nz,
 int[] subi,
 int[] subj,
 double[] valij,
 long[] 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` (`int[]`) – Dimensions of the symmetric matrixes. (input)
- `nz` (`long[]`) – Number of nonzeros for each matrix. (input)
- `subi` (`int[]`) – Row subscript in the triplets. (input)
- `subj` (`int[]`) – Column subscripts in the triplets. (input)

- `valij (double[])` – Values of each triplet. (input)
- `idx (long[])` – Unique index assigned to the inputted matrix that can be used for later reference. (output)

**Groups** *Problem data - semidefinite*

`Task.appendvars`

```
void appendvars (int 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 (int)` – Number of variables which should be appended. (input)

**Groups** *Problem data - linear part, Problem data - variables*

`Task.asyncgetresult`

```
int asyncgetresult
(string server,
 string port,
 string token,
 out rescode resp,
 out rescode trm)
```

```
void asyncgetresult
(string server,
 string port,
 string token,
 out int respavailable,
 out rescode resp,
 out rescode trm)
```

Request a response from a remote job. If successful, solver response, termination code and solutions are retrieved.

**Parameters**

- `server (string)` – Name or IP address of the solver server. (input)
- `port (string)` – Network port of the solver service. (input)
- `token (string)` – The task token. (input)
- `resp (rescode)` – Is the response code from the remote solver. (output)
- `trm (rescode)` – Is either *rescode.ok* or a termination response code. (output)
- `respavailable (int)` – Indicates if a remote response is available. If this is not true, `resp` and `trm` should be ignored. (output)

**Return** `(int)` – Indicates if a remote response is available. If this is not true, `resp` and `trm` should be ignored.

**Groups** *Remote optimization*

`Task.asyncoptimize`

```
string asyncoptimize
(string server,
 string port)
```

```
void asyncoptimize
(string server,
 string port,
 StringBuilder token)
```

Offload the optimization task to a solver server defined by **server:port**. The call will return immediately and not wait for the result.

If the string parameter *sparam.remote\_access\_token* is not blank, it will be passed to the server as authentication.

#### Parameters

- **server** (**string**) – Name or IP address of the solver server (input)
- **port** (**string**) – Network port of the solver service (input)
- **token** (**StringBuilder**) – Returns the task token (output)

**Return** (**string**) – Returns the task token

**Groups** *Remote optimization*

Task.asyncpoll

```
int asyncpoll
(string server,
 string port,
 string token,
 out rescode resp,
 out rescode trm)
```

```
void asyncpoll
(string server,
 string port,
 string token,
 out int respavailable,
 out rescode resp,
 out rescode trm)
```

Requests information about the status of the remote job.

#### Parameters

- **server** (**string**) – Name or IP address of the solver server (input)
- **port** (**string**) – Network port of the solver service (input)
- **token** (**string**) – The task token (input)
- **resp** (*rescode*) – Is the response code from the remote solver. (output)
- **trm** (*rescode*) – Is either *rescode.ok* or a termination response code. (output)
- **respavailable** (**int**) – Indicates if a remote response is available. If this is not true, **resp** and **trm** should be ignored. (output)

**Return** (**int**) – Indicates if a remote response is available. If this is not true, **resp** and **trm** should be ignored.

**Groups** *Remote optimization*

Task.asyncstop

```
void asyncstop
(string server,
 string port,
 string token)
```



Otherwise if `lower` is zero, then

$$\text{new upper bound} = \begin{cases} \infty, & \text{finite} = 0, \\ \text{value} & \text{otherwise.} \end{cases}$$

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` (`int`) – Index of the constraint for which the bounds should be changed. (input)
- `lower` (`int`) – If non-zero, then the lower bound is changed, otherwise the upper bound is changed. (input)
- `finite` (`int`) – If non-zero, then `value` is assumed to be finite. (input)
- `value` (`double`) – New value for the bound. (input)

**Groups** *Problem data - bounds, Problem data - constraints, Problem data - linear part*

`Task.chgvarbound`

```
void chgvarbound
(int j,
 int lower,
 int finite,
 double value)
```

Changes a bound for one variable.

If `lower` is non-zero, then the lower bound is changed as follows:

$$\text{new lower bound} = \begin{cases} -\infty, & \text{finite} = 0, \\ \text{value} & \text{otherwise.} \end{cases}$$

Otherwise if `lower` is zero, then

$$\text{new upper bound} = \begin{cases} \infty, & \text{finite} = 0, \\ \text{value} & \text{otherwise.} \end{cases}$$

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` (`int`) – Index of the variable for which the bounds should be changed. (input)
- `lower` (`int`) – If non-zero, then the lower bound is changed, otherwise the upper bound is changed. (input)
- `finite` (`int`) – If non-zero, then `value` is assumed to be finite. (input)
- `value` (`double`) – New value for the bound. (input)

**Groups** *Problem data - bounds, Problem data - variables, Problem data - linear part*

`Task.commitchanges`

```
void 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`



```
void generateconnames
(int[] subi,
 string fmt,
 int[] dims,
 long[] sp)
```

Generates systematic names for constraints.

**Parameters**

- subi (int[]) – Indexes of the constraints. (input)
- fmt (string) – The constraint name formatting string. (input)
- dims (int[]) – Dimensions in the shape. (input)
- sp (long[]) – Items that should be named. (input)

**Groups** *Names, Problem data - constraints, Problem data - linear part*

Task.generatevarnames

```
void generatevarnames
(int[] subj,
 string fmt,
 int[] dims,
 long[] sp)
```

Generates systematic names for variables.

**Parameters**

- subj (int[]) – Indexes of the variables. (input)
- fmt (string) – The variable name formatting string. (input)
- dims (int[]) – Dimensions in the shape. (input)
- sp (long[]) – Items that should be named. (input)

**Groups** *Names, Problem data - variables, Problem data - linear part*

Task.getacol

```
void getacol
(int j,
 out int nzj,
 int[] subj,
 double[] valj)
```

Obtains one column of  $A$  in a sparse format.

**Parameters**

- j (int) – Index of the column. (input)
- nzj (int) – Number of non-zeros in the column obtained. (output)
- subj (int[]) – Row indices of the non-zeros in the column obtained. (output)
- valj (double[]) – Numerical values in the column obtained. (output)

**Groups** *Problem data - linear part, Inspecting the task*

Task.getacolnumnz

```
int getacolnumnz (int i)
```

```
void getacolnumnz
(int i,
 out int nzj)
```

Obtains the number of non-zero elements in one column of  $A$ .

**Parameters**

- **i** (int) – Index of the column. (input)
- **nzj** (int) – Number of non-zeros in the  $j$ -th column of  $A$ . (output)

**Return** (int) – Number of non-zeros in the  $j$ -th column of  $A$ .

**Groups** *Problem data - linear part, Inspecting the task*

Task.getacolslice

```
void getacolslice
(int first,
 int last,
 ref long surp,
 long[] ptrb,
 long[] ptre,
 int[] sub,
 double[] val)
```

Obtains a sequence of columns from  $A$  in sparse format.

**Parameters**

- **first** (int) – Index of the first column in the sequence. (input)
- **last** (int) – Index of the last column in the sequence **plus one**. (input)
- **surp** (long) – Surplus of subscript and coefficient arrays. The required entries are stored sequentially in **sub** and **val** starting from position **surp** away from the end of the arrays. Upon return **surp** will be decremented by the total number of non-zeros written. (input/output)
- **ptrb** (long[]) – **ptrb[t]** is an index pointing to the first element in the  $t$ -th column obtained. (output)
- **ptre** (long[]) – **ptre[t]** is an index pointing to the last element plus one in the  $t$ -th column obtained. (output)
- **sub** (int[]) – Contains the row subscripts. (output)
- **val** (double[]) – Contains the coefficient values. (output)

**Groups** *Problem data - linear part, Inspecting the task*

Task.getacolslicenumnz

```
long getacolslicenumnz
(int first,
 int last)
```

```
void getacolslicenumnz
(int first,
 int last,
 out long numnz)
```

Obtains the number of non-zeros in a slice of columns of  $A$ .

**Parameters**

- **first** (int) – Index of the first column in the sequence. (input)
- **last** (int) – Index of the last column **plus one** in the sequence. (input)
- **numnz** (long) – Number of non-zeros in the slice. (output)

**Return** (long) – Number of non-zeros in the slice.

**Groups** *Problem data - linear part, Inspecting the task*

### Task.getacolslicetrip

```
void getacolslicetrip
(int first,
 int last,
 ref long surp,
 int[] subi,
 int[] subj,
 double[] val)
```

Obtains a sequence of columns from  $A$  in sparse triplet format. The function returns the content of all columns whose index  $j$  satisfies  $\text{first} \leq j < \text{last}$ . The triplets corresponding to nonzero entries are stored in the arrays `subi`, `subj` and `val`.

#### Parameters

- `first` (`int`) – Index of the first column in the sequence. (input)
- `last` (`int`) – Index of the last column in the sequence **plus one**. (input)
- `surp` (`long`) – Surplus of subscript and coefficient arrays. The required entries are stored sequentially in `subi`, `subj` and `val` starting from position `surp` away from the end of the arrays. On return `surp` will be decremented by the total number of non-zeros written. (input/output)
- `subi` (`int[]`) – Constraint subscripts. (output)
- `subj` (`int[]`) – Column subscripts. (output)
- `val` (`double[]`) – Values. (output)

**Groups** *Problem data - linear part, Inspecting the task*

### Task.getaij

```
double getaij
(int i,
 int j)
```

```
void getaij
(int i,
 int j,
 out double aij)
```

Obtains a single coefficient in  $A$ .

#### Parameters

- `i` (`int`) – Row index of the coefficient to be returned. (input)
- `j` (`int`) – Column index of the coefficient to be returned. (input)
- `aij` (`double`) – The required coefficient  $a_{i,j}$ . (output)

**Return** (`double`) – The required coefficient  $a_{i,j}$ .

**Groups** *Problem data - linear part, Inspecting the task*

### Task.getapieceenumnz

```
int getapieceenumnz
(int firsti,
 int lasti,
 int firstj,
 int lastj)
```



Task.getarowslice

```
void getarowslice
(int first,
 int last,
 ref long surp,
 long[] ptrb,
 long[] ptre,
 int[] sub,
 double[] val)
```

Obtains a sequence of rows from  $A$  in sparse format.

#### Parameters

- **first** (int) – Index of the first row in the sequence. (input)
- **last** (int) – Index of the last row in the sequence **plus one**. (input)
- **surp** (long) – Surplus of subscript and coefficient arrays. The required entries are stored sequentially in **sub** and **val** starting from position **surp** away from the end of the arrays. Upon return **surp** will be decremented by the total number of non-zeros written. (input/output)
- **ptrb** (long[]) – **ptrb[t]** is an index pointing to the first element in the  $t$ -th row obtained. (output)
- **ptre** (long[]) – **ptre[t]** is an index pointing to the last element plus one in the  $t$ -th row obtained. (output)
- **sub** (int[]) – Contains the column subscripts. (output)
- **val** (double[]) – Contains the coefficient values. (output)

**Groups** *Problem data - linear part, Inspecting the task*

Task.getarowslicenumnz

```
long getarowslicenumnz
(int first,
 int last)
```

```
void getarowslicenumnz
(int first,
 int last,
 out long numnz)
```

Obtains the number of non-zeros in a slice of rows of  $A$ .

#### Parameters

- **first** (int) – Index of the first row in the sequence. (input)
- **last** (int) – Index of the last row **plus one** in the sequence. (input)
- **numnz** (long) – Number of non-zeros in the slice. (output)

**Return** (long) – Number of non-zeros in the slice.

**Groups** *Problem data - linear part, Inspecting the task*

Task.getarowslicetrip

```
void getarowslicetrip
(int first,
 int last,
 ref long surp,
```

(continues on next page)

```
int[] subi,
int[] subj,
double[] 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`  $\leq i <$  `last`. The triplets corresponding to nonzero entries are stored in the arrays `subi`, `subj` and `val`.

#### Parameters

- `first` (int) – Index of the first row in the sequence. (input)
- `last` (int) – Index of the last row in the sequence **plus one**. (input)
- `surp` (long) – Surplus of subscript and coefficient arrays. The required entries are stored sequentially in `subi`, `subj` and `val` starting from position `surp` away from the end of the arrays. On return `surp` will be decremented by the total number of non-zeros written. (input/output)
- `subi` (int[]) – Constraint subscripts. (output)
- `subj` (int[]) – Column subscripts. (output)
- `val` (double[]) – Values. (output)

**Groups** *Problem data - linear part, Inspecting the task*

`Task.getatruncatetol`

```
void getatruncatetol (double[] tolzero)
```

Obtains the tolerance value set with *Task.putatruncatetol*.

**Parameters** `tolzero` (double[]) – All elements  $|a_{i,j}|$  less than this tolerance is truncated to zero. (output)

**Groups** *Parameters, Problem data - linear part*

`Task.getbarablocktriplet`

```
long getbarablocktriplet
(int[] subi,
int[] subj,
int[] subk,
int[] subl,
double[] valijkl)
```

```
void getbarablocktriplet
(out long num,
int[] subi,
int[] subj,
int[] subk,
int[] subl,
double[] valijkl)
```

Obtains  $\bar{A}$  in block triplet form.

#### Parameters

- `subi` (int[]) – Constraint index. (output)
- `subj` (int[]) – Symmetric matrix variable index. (output)
- `subk` (int[]) – Block row index. (output)
- `subl` (int[]) – Block column index. (output)
- `valijkl` (double[]) – The numerical value associated with each block triplet. (output)

- **num (long)** – Number of elements in the block triplet form. (output)

**Return (long)** – Number of elements in the block triplet form.

**Groups** *Problem data - semidefinite, Inspecting the task*

Task.getbaraidx

```
long getbaraidx
(long idx,
 out int i,
 out int j,
 long[] sub,
 double[] weights)
```

```
void getbaraidx
(long idx,
 out int i,
 out int j,
 out long num,
 long[] sub,
 double[] weights)
```

Obtains information about an element in  $\bar{A}$ . Since  $\bar{A}$  is a sparse matrix of symmetric matrices, only the nonzero elements in  $\bar{A}$  are stored in order to save space. Now  $\bar{A}$  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  $\bar{A}$ .

Please observe if one element of  $\bar{A}$  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 (long)** – Position of the element in the vectorized form. (input)
- **i (int)** – Row index of the element at position **idx**. (output)
- **j (int)** – Column index of the element at position **idx**. (output)
- **sub (long[])** – A list indexes of the elements from symmetric matrix storage that appear in the weighted sum. (output)
- **weights (double[])** – The weights associated with each term in the weighted sum. (output)
- **num (long)** – Number of terms in weighted sum that forms the element. (output)

**Return (long)** – Number of terms in weighted sum that forms the element.

**Groups** *Problem data - semidefinite, Inspecting the task*

Task.getbaraidxij

```
void getbaraidxij
(long idx,
 out int i,
 out int j)
```

Obtains information about an element in  $\bar{A}$ . Since  $\bar{A}$  is a sparse matrix of symmetric matrices, only the nonzero elements in  $\bar{A}$  are stored in order to save space. Now  $\bar{A}$  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  $\bar{A}$ .

Please note that if one element of  $\bar{A}$  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 (long)** – Position of the element in the vectorized form. (input)

- `i` (`int`) – Row index of the element at position `idx`. (output)
- `j` (`int`) – Column index of the element at position `idx`. (output)

**Groups** *Problem data - semidefnite, Inspecting the task*

`Task.getbaraidxinfo`

```
long getbaraidxinfo (long idx)
```

```
void getbaraidxinfo
(long idx,
 out long num)
```

Each nonzero element in  $\bar{A}_{ij}$  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` (`long`) – The internal position of the element for which information should be obtained. (input)
- `num` (`long`) – Number of terms in the weighted sum that form the specified element in  $\bar{A}$ . (output)

**Return** (`long`) – Number of terms in the weighted sum that form the specified element in  $\bar{A}$ .

**Groups** *Problem data - semidefnite, Inspecting the task*

`Task.getbarasparsity`

```
void getbarasparsity
(out long numnz,
 long[] idxij)
```

The matrix  $\bar{A}$  is assumed to be a sparse matrix of symmetric matrices. This implies that many of the elements in  $\bar{A}$  are likely to be zero matrices. Therefore, in order to save space, only nonzero elements in  $\bar{A}$  are stored on vectorized form. This function is used to obtain the sparsity pattern of  $\bar{A}$  and the position of each nonzero element in the vectorized form of  $\bar{A}$ . From the index detailed information about each nonzero  $\bar{A}_{i,j}$  can be obtained using [Task.getbaraidxinfo](#) and [Task.getbaraidx](#).

**Parameters**

- `numnz` (`long`) – Number of nonzero elements in  $\bar{A}$ . (output)
- `idxij` (`long[]`) – Position of each nonzero element in the vectorized form of  $\bar{A}$ . (output)

**Groups** *Problem data - semidefnite, Inspecting the task*

`Task.getbarcblocktriplet`

```
long getbarcblocktriplet
(int[] subj,
 int[] subk,
 int[] subl,
 double[] valjkl)
```

```
void getbarcblocktriplet
(out long num,
 int[] subj,
```

(continues on next page)

```
int[] subk,
int[] subl,
double[] valjkl)
```

Obtains  $\overline{C}$  in block triplet form.

#### Parameters

- `subj (int[])` – Symmetric matrix variable index. (output)
- `subk (int[])` – Block row index. (output)
- `subl (int[])` – Block column index. (output)
- `valjkl (double[])` – The numerical value associated with each block triplet. (output)
- `num (long)` – Number of elements in the block triplet form. (output)

**Return** (long) – Number of elements in the block triplet form.

**Groups** *Problem data - semidefinite, Inspecting the task*

`Task.getbarcidx`

```
void getbarcidx
(long idx,
 out int j,
 out long num,
 long[] sub,
 double[] weights)
```

Obtains information about an element in  $\overline{C}$ .

#### Parameters

- `idx (long)` – Index of the element for which information should be obtained. (input)
- `j (int)` – Row index in  $\overline{C}$ . (output)
- `num (long)` – Number of terms in the weighted sum. (output)
- `sub (long[])` – Elements appearing the weighted sum. (output)
- `weights (double[])` – Weights of terms in the weighted sum. (output)

**Groups** *Problem data - semidefinite, Inspecting the task*

`Task.getbarcidxinfo`

```
long getbarcidxinfo (long idx)
```

```
void getbarcidxinfo
(long idx,
 out long num)
```

Obtains the number of terms in the weighted sum that forms a particular element in  $\overline{C}$ .

#### Parameters

- `idx (long)` – Index of the element for which information should be obtained. The value is an index of a symmetric sparse variable. (input)
- `num (long)` – Number of terms that appear in the weighted sum that forms the requested element. (output)

**Return** (long) – Number of terms that appear in the weighted sum that forms the requested element.

**Groups** *Problem data - semidefinite, Inspecting the task*

Task.getbarcidxj

```
void getbarcidxj
(long idx,
 out int j)
```

Obtains the row index of an element in  $\overline{C}$ .

**Parameters**

- `idx` (`long`) – Index of the element for which information should be obtained. (input)
- `j` (`int`) – Row index in  $\overline{C}$ . (output)

**Groups** *Problem data - semidefinite, Inspecting the task*

Task.getbarcsparsity

```
void getbarcsparsity
(out long numnz,
 long[] 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**

- `numnz` (`long`) – Number of nonzero elements in  $\overline{C}$ . (output)
- `idxj` (`long[]`) – Internal positions of the nonzero elements in  $\overline{C}$ . (output)

**Groups** *Problem data - semidefinite, Inspecting the task*

Task.getbarsj

```
void getbarsj
(soltype whichsol,
 int j,
 double[] 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` (*`soltype`*) – Selects a solution. (input)
- `j` (`int`) – Index of the semidefinite variable. (input)
- `barsj` (`double[]`) – Value of  $\overline{S}_j$ . (output)

**Groups** *Solution - semidefinite*

Task.getbarsslice

```
void getbarsslice
(soltype whichsol,
 int first,
 int last,
 long slicesize,
 double[] barsslice)
```





**Parameters** `c (double[])` – Linear terms of the objective as a dense vector. The length is the number of variables. (output)

**Groups** *Problem data - linear part, Inspecting the task, Problem data - variables*

`Task.getcfix`

```
double getcfix ()
```

```
void getcfix (out double cfix)
```

Obtains the fixed term in the objective.

**Parameters** `cfix (double)` – Fixed term in the objective. (output)

**Return** `(double)` – Fixed term in the objective.

**Groups** *Problem data - linear part, Inspecting the task*

`Task.getcj`

```
void getcj  
(int j,  
 out double cj)
```

Obtains one coefficient of  $c$ .

**Parameters**

- `j (int)` – Index of the variable for which the  $c$  coefficient should be obtained. (input)
- `cj (double)` – The value of  $c_j$ . (output)

**Groups** *Problem data - linear part, Inspecting the task, Problem data - variables*

`Task.getclist`

```
void getclist  
(int[] subj,  
 double[] c)
```

Obtains a sequence of elements in  $c$ .

**Parameters**

- `subj (int[])` – A list of variable indexes. (input)
- `c (double[])` – Linear terms of the requested list of the objective as a dense vector. (output)

**Groups** *Inspecting the task, Problem data - linear part*

`Task.getconbound`

```
void getconbound  
(int i,  
 out boundkey bk,  
 out double bl,  
 out double bu)
```

Obtains bound information for one constraint.

**Parameters**

- `i (int)` – Index of the constraint for which the bound information should be obtained. (input)

- **bk** (*boundkey*) – Bound keys. (output)
- **bl** (*double*) – Values for lower bounds. (output)
- **bu** (*double*) – Values for upper bounds. (output)

**Groups** *Problem data - linear part, Inspecting the task, Problem data - bounds, Problem data - constraints*

**Task.getconboundslice**

```
void getconboundslice
(int first,
 int last,
 boundkey[] bk,
 double[] bl,
 double[] bu)
```

Obtains bounds information for a slice of the constraints.

**Parameters**

- **first** (*int*) – First index in the sequence. (input)
- **last** (*int*) – Last index plus 1 in the sequence. (input)
- **bk** (*boundkey* []) – Bound keys. (output)
- **bl** (*double* []) – Values for lower bounds. (output)
- **bu** (*double* []) – Values for upper bounds. (output)

**Groups** *Problem data - linear part, Inspecting the task, Problem data - bounds, Problem data - constraints*

**Task.getcone**

```
void getcone
(int k,
 out conetype ct,
 out double coneapar,
 out int nummem,
 int[] submem)
```

Obtains a cone.

**Parameters**

- **k** (*int*) – Index of the cone. (input)
- **ct** (*conetype*) – Specifies the type of the cone. (output)
- **coneapar** (*double*) – For the power cone it denotes the exponent alpha. For other cone types it is unused and can be set to 0. (output)
- **nummem** (*int*) – Number of member variables in the cone. (output)
- **submem** (*int* []) – Variable subscripts of the members in the cone. (output)

**Groups** *Inspecting the task, Problem data - cones*

**Task.getconeinfo**

```
void getconeinfo
(int k,
 out conetype ct,
 out double coneapar,
 out int nummem)
```

Obtains information about a cone.

### Parameters

- **k** (**int**) – Index of the cone. (input)
- **ct** (*conetype*) – Specifies the type of the cone. (output)
- **conepar** (**double**) – For the power cone it denotes the exponent alpha. For other cone types it is unused and can be set to 0. (output)
- **nummem** (**int**) – Number of member variables in the cone. (output)

**Groups** *Inspecting the task, Problem data - cones*

Task.getconename

```
string getconename (int i)
```

```
void getconename  
(int i,  
StringBuilder name)
```

Obtains the name of a cone.

### Parameters

- **i** (**int**) – Index of the cone. (input)
- **name** (**StringBuilder**) – The required name. (output)

**Return** (**string**) – The required name.

**Groups** *Names, Problem data - cones, Inspecting the task*

Task.getconenameindex

```
int getconenameindex  
(string somename,  
out int asgn)
```

```
void getconenameindex  
(string somename,  
out int asgn,  
out int index)
```

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** (**string**) – The name which should be checked. (input)
- **asgn** (**int**) – Is non-zero if the name **somename** is assigned to some cone. (output)
- **index** (**int**) – If the name **somename** is assigned to some cone, then **index** is the index of the cone. (output)

**Return** (**int**) – If the name **somename** is assigned to some cone, then **index** is the index of the cone.

**Groups** *Names, Problem data - cones, Inspecting the task*

Task.getconenamelen

```
int getconenamelen (int i)
```

```
void getconenamelen  
(int i,  
out int len)
```

Obtains the length of the name of a cone.

**Parameters**

- **i** (**int**) – Index of the cone. (input)
- **len** (**int**) – Returns the length of the indicated name. (output)

**Return** (**int**) – Returns the length of the indicated name.

**Groups** *Names, Problem data - cones, Inspecting the task*

Task.getconname

```
string getconname (int i)
```

```
void getconname  
(int i,  
  StringBuilder name)
```

Obtains the name of a constraint.

**Parameters**

- **i** (**int**) – Index of the constraint. (input)
- **name** (**StringBuilder**) – The required name. (output)

**Return** (**string**) – The required name.

**Groups** *Names, Problem data - linear part, Problem data - constraints, Inspecting the task*

Task.getconnameindex

```
int getconnameindex  
(string somename,  
  out int asgn)
```

```
void getconnameindex  
(string somename,  
  out int asgn,  
  out int index)
```

Checks whether the name **somename** has been assigned to any constraint. If so, the index of the constraint is reported.

**Parameters**

- **somename** (**string**) – The name which should be checked. (input)
- **asgn** (**int**) – Is non-zero if the name **somename** is assigned to some constraint. (output)
- **index** (**int**) – If the name **somename** is assigned to a constraint, then **index** is the index of the constraint. (output)

**Return** (**int**) – 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

```
int getconnamelen (int i)
```













**Groups** *Inspecting the task, Problem data - linear part*

Task.getmaxnumbarvar

```
int getmaxnumbarvar ()
```

```
void getmaxnumbarvar (out int maxnumbarvar)
```

Obtains maximum number of symmetric matrix variables for which space is currently preallocated.

**Parameters** maxnumbarvar (int) – Maximum number of symmetric matrix variables for which space is currently preallocated. (output)

**Return** (int) – Maximum number of symmetric matrix variables for which space is currently preallocated.

**Groups** *Inspecting the task, Problem data - semidefinite*

Task.getmaxnumcon

```
void getmaxnumcon (out int 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.

**Parameters** maxnumcon (int) – Number of preallocated constraints in the optimization task. (output)

**Groups** *Inspecting the task, Problem data - linear part, Problem data - constraints*

Task.getmaxnumcone

```
void getmaxnumcone (out int maxnumcone)
```

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.

**Parameters** maxnumcone (int) – Number of preallocated conic constraints in the optimization task. (output)

**Groups** *Inspecting the task, Problem data - cones*

Task.getmaxnumqnz

```
void getmaxnumqnz (out long 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  $Q$ .

**Parameters** maxnumqnz (long) – Number of non-zero elements preallocated in quadratic coefficient matrices. (output)

**Groups** *Inspecting the task, Problem data - quadratic part*

Task.getmaxnumvar

```
void getmaxnumvar (out int 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.

**Parameters** maxnumvar (int) – Number of preallocated variables in the optimization task. (output)

**Groups** *Inspecting the task, Problem data - linear part, Problem data - variables*

**Task.getmemusage**

```
void getmemusage
(out long meminuse,
 out long maxmemuse)
```

Obtains information about the amount of memory used by a task.

**Parameters**

- **meminuse** (long) – Amount of memory currently used by the **task**. (output)
- **maxmemuse** (long) – Maximum amount of memory used by the **task** until now. (output)

**Groups** *System, memory and debugging*

**Task.getnumanz**

```
int getnumanz ()
```

```
void getnumanz (out int numanz)
```

Obtains the number of non-zeros in  $A$ .

**Parameters** **numanz** (int) – Number of non-zero elements in the linear constraint matrix. (output)

**Return** (int) – Number of non-zero elements in the linear constraint matrix.

**Groups** *Inspecting the task, Problem data - linear part*

**Task.getnumanz64**

```
long getnumanz64 ()
```

```
void getnumanz64 (out long numanz)
```

Obtains the number of non-zeros in  $A$ .

**Parameters** **numanz** (long) – Number of non-zero elements in the linear constraint matrix. (output)

**Return** (long) – Number of non-zero elements in the linear constraint matrix.

**Groups** *Inspecting the task, Problem data - linear part*

**Task.getnumbarablocktriplets**

```
long getnumbarablocktriplets ()
```

```
void getnumbarablocktriplets (out long num)
```

Obtains an upper bound on the number of elements in the block triplet form of  $\overline{A}$ .

**Parameters** **num** (long) – An upper bound on the number of elements in the block triplet form of  $\overline{A}$ . (output)

**Return** (long) – 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

```
long getnumbaranz ()
```

```
void getnumbaranz (out long nz)
```

Get the number of nonzero elements in  $\bar{A}$ .

**Parameters** `nz` (`long`) – The number of nonzero block elements in  $\bar{A}$  i.e. the number of  $\bar{A}_{ij}$  elements that are nonzero. (output)

**Return** (`long`) – The number of nonzero block elements in  $\bar{A}$  i.e. the number of  $\bar{A}_{ij}$  elements that are nonzero.

**Groups** *Problem data - semidefinite, Inspecting the task*

Task.getnumbarcblocktriplets

```
long getnumbarcblocktriplets ()
```

```
void getnumbarcblocktriplets (out long num)
```

Obtains an upper bound on the number of elements in the block triplet form of  $\bar{C}$ .

**Parameters** `num` (`long`) – An upper bound on the number of elements in the block triplet form of  $\bar{C}$ . (output)

**Return** (`long`) – An upper bound on the number of elements in the block triplet form of  $\bar{C}$ .

**Groups** *Problem data - semidefinite, Inspecting the task*

Task.getnumbarcnz

```
long getnumbarcnz ()
```

```
void getnumbarcnz (out long nz)
```

Obtains the number of nonzero elements in  $\bar{C}$ .

**Parameters** `nz` (`long`) – The number of nonzeros in  $\bar{C}$  i.e. the number of elements  $\bar{C}_j$  that are nonzero. (output)

**Return** (`long`) – The number of nonzeros in  $\bar{C}$  i.e. the number of elements  $\bar{C}_j$  that are nonzero.

**Groups** *Problem data - semidefinite, Inspecting the task*

Task.getnumbarvar

```
int getnumbarvar ()
```

```
void getnumbarvar (out int numbarvar)
```

Obtains the number of semidefinite variables.

**Parameters** `numbarvar` (`int`) – Number of semidefinite variables in the problem. (output)

**Return** (`int`) – Number of semidefinite variables in the problem.

**Groups** *Inspecting the task, Problem data - semidefinite*

Task.getnumcon

```
int getnumcon ()
```

```
void getnumcon (out int numcon)
```

Obtains the number of constraints.

**Parameters** numcon (int) – Number of constraints. (output)

**Return** (int) – Number of constraints.

**Groups** *Problem data - linear part, Problem data - constraints, Inspecting the task*

Task.getnumcone

```
int getnumcone ()
```

```
void getnumcone (out int numcone)
```

Obtains the number of cones.

**Parameters** numcone (int) – Number of conic constraints. (output)

**Return** (int) – Number of conic constraints.

**Groups** *Problem data - cones, Inspecting the task*

Task.getnumconemem

```
void getnumconemem  
  (int k,  
   out int nummem)
```

Obtains the number of members in a cone.

**Parameters**

- k (int) – Index of the cone. (input)
- nummem (int) – Number of member variables in the cone. (output)

**Groups** *Problem data - cones, Inspecting the task*

Task.getnumintvar

```
void getnumintvar (out int numintvar)
```

Obtains the number of integer-constrained variables.

**Parameters** numintvar (int) – Number of integer variables. (output)

**Groups** *Inspecting the task, Problem data - variables*

Task.getnumparam

```
void getnumparam  
  (parametertype partype,  
   out int numparam)
```

Obtains the number of parameters of a given type.

**Parameters**

- partype (*parametertype*) – Parameter type. (input)









```
void getpviolvar
(soltype whichsol,
 int[] sub,
 double[] 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** (*soltype*) – Selects a solution. (input)
- **sub** (*int[]*) – An array of indexes of  $x$  variables. (input)
- **viol** (*double[]*) – **viol[k]** is the violation associated with the solution for the variable  $x_{\text{sub}[k]}$ . (output)

**Groups** *Solution information*

**Task.**getqconk

```
long getqconk
(int k,
 ref long qcsurp,
 int[] qcsubi,
 int[] qcsubj,
 double[] qcval)
```

```
void getqconk
(int k,
 ref long qcsurp,
 out long numqcnz,
 int[] qcsubi,
 int[] qcsubj,
 double[] qcval)
```

Obtains all the quadratic terms in a constraint. The quadratic terms are stored sequentially in **qcsubi**, **qcsubj**, and **qcval**.

#### Parameters

- **k** (*int*) – Which constraint. (input)
- **qcsurp** (*long*) – Surplus of subscript and coefficient arrays. The required entries are stored sequentially in **qcsubi**, **qcsubj** and **qcval** starting from position **qcsurp** away from the end of the arrays. On return **qcsurp** will be decremented by the total number of non-zeros written. (input/output)
- **qcsubi** (*int[]*) – Row subscripts for quadratic constraint matrix. (output)
- **qcsubj** (*int[]*) – Column subscripts for quadratic constraint matrix. (output)
- **qcval** (*double[]*) – Quadratic constraint coefficient values. (output)
- **numqcnz** (*long*) – Number of quadratic terms. (output)

**Return** (*long*) – Number of quadratic terms.

**Groups** *Inspecting the task, Problem data - quadratic part, Problem data - constraints*

**Task.**getqobj

```

void getqobj
(ref long qosurp,
 out long numqonz,
 int[] qosubi,
 int[] qosubj,
 double[] qoval)

```

Obtains the quadratic terms in the objective. The required quadratic terms are stored sequentially in `qosubi`, `qosubj`, and `qoval`.

#### Parameters

- `qosurp` (`long`) – Surplus of subscript and coefficient arrays. The required entries are stored sequentially in `qosubi`, `qosubj` and `qoval` starting from position `qosurp` away from the end of the arrays. On return `qosurp` will be decremented by the total number of non-zeros written. (input/output)
- `numqonz` (`long`) – Number of non-zero elements in the quadratic objective terms. (output)
- `qosubi` (`int[]`) – Row subscripts for quadratic objective coefficients. (output)
- `qosubj` (`int[]`) – Column subscripts for quadratic objective coefficients. (output)
- `qoval` (`double[]`) – Quadratic objective coefficient values. (output)

**Groups** *Inspecting the task, Problem data - quadratic part*

`Task.getqobjij`

```

void getqobjij
(int i,
 int j,
 out double qoij)

```

Obtains one coefficient  $q_{ij}^o$  in the quadratic term of the objective.

#### Parameters

- `i` (`int`) – Row index of the coefficient. (input)
- `j` (`int`) – Column index of coefficient. (input)
- `qoij` (`double`) – The required coefficient. (output)

**Groups** *Inspecting the task, Problem data - quadratic part*

`Task.getreducedcosts`

```

void getreducedcosts
(soltype whichsol,
 int first,
 int last,
 double[] redcosts)

```

Computes the reduced costs for a slice of variables and returns them in the array `redcosts` i.e.

$$\text{redcosts}[j - \text{first}] = (s_l^x)_j - (s_u^x)_j, \quad j = \text{first}, \dots, \text{last} - 1 \quad (15.2)$$

#### Parameters

- `whichsol` (`soltype`) – Selects a solution. (input)
- `first` (`int`) – The index of the first variable in the sequence. (input)
- `last` (`int`) – The index of the last variable in the sequence plus 1. (input)
- `redcosts` (`double[]`) – The reduced costs for the required slice of variables. (output)

## Groups *Solution - dual*

Task.getskc

```
void getskc
(soltype whichsol,
 stakekey[] skc)
```

Obtains the status keys for the constraints.

### Parameters

- `whichsol` (*soltype*) – Selects a solution. (input)
- `skc` (*stakekey*[]) – Status keys for the constraints. (output)

## Groups *Solution information*

Task.getskcslice

```
void getskcslice
(soltype whichsol,
 int first,
 int last,
 stakekey[] skc)
```

Obtains the status keys for a slice of the constraints.

### Parameters

- `whichsol` (*soltype*) – Selects a solution. (input)
- `first` (int) – First index in the sequence. (input)
- `last` (int) – Last index plus 1 in the sequence. (input)
- `skc` (*stakekey*[]) – Status keys for the constraints. (output)

## Groups *Solution information*

Task.getskn

```
void getskn
(soltype whichsol,
 stakekey[] skn)
```

Obtains the status keys for the conic constraints.

### Parameters

- `whichsol` (*soltype*) – Selects a solution. (input)
- `skn` (*stakekey*[]) – Status keys for the conic constraints. (output)

## Groups *Solution information*

Task.getskx

```
void getskx
(soltype whichsol,
 stakekey[] skx)
```

Obtains the status keys for the scalar variables.

### Parameters

- `whichsol` (*soltype*) – Selects a solution. (input)
- `skx` (*stakekey*[]) – Status keys for the variables. (output)

## Groups *Solution information*

Task.getskxslice

```
void getskxslice
(soltype whichsol,
 int first,
 int last,
 stakey[] skx)
```

Obtains the status keys for a slice of the scalar variables.

### Parameters

- **whichsol** (*soltype*) – Selects a solution. (input)
- **first** (int) – First index in the sequence. (input)
- **last** (int) – Last index plus 1 in the sequence. (input)
- **skx** (*stakey*[]) – Status keys for the variables. (output)

## Groups *Solution information*

Task.getslc

```
void getslc
(soltype whichsol,
 double[] slc)
```

Obtains the  $s_l^c$  vector for a solution.

### Parameters

- **whichsol** (*soltype*) – Selects a solution. (input)
- **slc** (double[]) – Dual variables corresponding to the lower bounds on the constraints. (output)

## Groups *Solution - dual*

Task.getslcslice

```
void getslcslice
(soltype whichsol,
 int first,
 int last,
 double[] slc)
```

Obtains a slice of the  $s_l^c$  vector for a solution.

### Parameters

- **whichsol** (*soltype*) – Selects a solution. (input)
- **first** (int) – First index in the sequence. (input)
- **last** (int) – Last index plus 1 in the sequence. (input)
- **slc** (double[]) – Dual variables corresponding to the lower bounds on the constraints. (output)

## Groups *Solution - dual*

Task.getslx

```
void getslx
(soltype whichsol,
 double[] slx)
```

Obtains the  $s_l^x$  vector for a solution.

**Parameters**

- `whichsol` (*[soltype](#)*) – Selects a solution. (input)
- `slx` (`double[]`) – Dual variables corresponding to the lower bounds on the variables. (output)

**Groups** *[Solution](#) - [dual](#)*

`Task.getslxslice`

```
void getslxslice
(soltype whichsol,
 int first,
 int last,
 double[] slx)
```

Obtains a slice of the  $s_l^x$  vector for a solution.

**Parameters**

- `whichsol` (*[soltype](#)*) – Selects a solution. (input)
- `first` (`int`) – First index in the sequence. (input)
- `last` (`int`) – Last index plus 1 in the sequence. (input)
- `slx` (`double[]`) – Dual variables corresponding to the lower bounds on the variables. (output)

**Groups** *[Solution](#) - [dual](#)*

`Task.getsnx`

```
void getsnx
(soltype whichsol,
 double[] snx)
```

Obtains the  $s_n^x$  vector for a solution.

**Parameters**

- `whichsol` (*[soltype](#)*) – Selects a solution. (input)
- `snx` (`double[]`) – Dual variables corresponding to the conic constraints on the variables. (output)

**Groups** *[Solution](#) - [dual](#)*

`Task.getsnxslice`

```
void getsnxslice
(soltype whichsol,
 int first,
 int last,
 double[] snx)
```

Obtains a slice of the  $s_n^x$  vector for a solution.

**Parameters**

- `whichsol` (*[soltype](#)*) – Selects a solution. (input)
- `first` (`int`) – First index in the sequence. (input)
- `last` (`int`) – Last index plus 1 in the sequence. (input)
- `snx` (`double[]`) – Dual variables corresponding to the conic constraints on the variables. (output)

## Groups *Solution - dual*

Task.getsolsta

```
solsta getsolsta (soltype whichsol)
```

```
void getsolsta
(soltype whichsol,
 out solsta solsta)
```

Obtains the solution status.

### Parameters

- **whichsol** (*soltype*) – Selects a solution. (input)
- **solsta** (*solsta*) – Solution status. (output)

**Return** (*solsta*) – Solution status.

**Groups** *Solution information*

Task.getsolution

```
void getsolution
(soltype whichsol,
 out prosta prosta,
 out solsta solsta,
 stakey[] skc,
 stakey[] skx,
 stakey[] skn,
 double[] xc,
 double[] xx,
 double[] y,
 double[] slc,
 double[] suc,
 double[] slx,
 double[] sux,
 double[] snx)
```

Obtains the complete solution.

Consider the case of linear programming. The primal problem is given by

$$\begin{array}{ll} \text{minimize} & c^T x + c^f \\ \text{subject to} & l^c \leq Ax \leq u^c, \\ & l^x \leq x \leq u^x. \end{array}$$

and the corresponding dual problem is

$$\begin{array}{ll} \text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c \\ & + (l^x)^T s_l^x - (u^x)^T s_u^x + c^f \\ \text{subject to} & A^T y + s_l^x - s_u^x = c, \\ & -y + s_l^c - s_u^c = 0, \\ & s_l^c, s_u^c, s_l^x, s_u^x \geq 0. \end{array}$$

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:

$$\begin{array}{ll} \text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c \\ & + (l^x)^T s_l^x - (u^x)^T s_u^x + c^f \\ \text{subject to} & A^T y + s_l^x - s_u^x + s_n^x = c, \\ & -y + s_l^c - s_u^c = 0, \\ & s_l^c, s_u^c, s_l^x, s_u^x \geq 0, \\ & s_n^x \in \mathcal{K}^* \end{array}$$

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** (*soltype*) – Selects a solution. (input)
- **prosta** (*prosta*) – Problem status. (output)
- **solsta** (*solsta*) – Solution status. (output)
- **skc** (*stakey*[]) – Status keys for the constraints. (output)
- **skx** (*stakey*[]) – Status keys for the variables. (output)
- **skn** (*stakey*[]) – Status keys for the conic constraints. (output)
- **xc** (*double*[]) – Primal constraint solution. (output)
- **xx** (*double*[]) – Primal variable solution. (output)
- **y** (*double*[]) – Vector of dual variables corresponding to the constraints. (output)
- **slc** (*double*[]) – Dual variables corresponding to the lower bounds on the constraints. (output)
- **suc** (*double*[]) – Dual variables corresponding to the upper bounds on the constraints. (output)
- **slx** (*double*[]) – Dual variables corresponding to the lower bounds on the variables. (output)
- **sux** (*double*[]) – Dual variables corresponding to the upper bounds on the variables. (output)
- **snx** (*double*[]) – Dual variables corresponding to the conic constraints on the variables. (output)

**Groups** *Solution information, Solution - primal, Solution - dual*

**Task.getsolutioninfo**

```

void getsolutioninfo
(soltype whichsol,
 out double pobj,
 out double pviolcon,
 out double pviolvar,
 out double pviolbarvar,
 out double pviolcone,
 out double pviolitg,
 out double dobj,
 out double dviolcon,
 out double dviolvar,
 out double dviolbarvar,
 out double dviolcone)

```

Obtains information about a solution.

#### Parameters

- `whichsol` (*soltype*) – Selects a solution. (input)
- `pobj` (double) – The primal objective value as computed by *Task.getprimalobj*. (output)
- `pviolcon` (double) – Maximal primal violation of the solution associated with the  $x^c$  variables where the violations are computed by *Task.getpviolcon*. (output)
- `pviolvar` (double) – Maximal primal violation of the solution for the  $x$  variables where the violations are computed by *Task.getpviolvar*. (output)
- `pviolbarvar` (double) – Maximal primal violation of solution for the  $\bar{X}$  variables where the violations are computed by *Task.getpviolbarvar*. (output)
- `pviolcone` (double) – Maximal primal violation of solution for the conic constraints where the violations are computed by *Task.getpviolcones*. (output)
- `pviolitg` (double) – 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. (output)
- `dobj` (double) – Dual objective value as computed by *Task.getdualobj*. (output)
- `dviolcon` (double) – Maximal violation of the dual solution associated with the  $x^c$  variable as computed by *Task.getdviolcon*. (output)
- `dviolvar` (double) – Maximal violation of the dual solution associated with the  $x$  variable as computed by *Task.getdviolvar*. (output)
- `dviolbarvar` (double) – Maximal violation of the dual solution associated with the  $\bar{S}$  variable as computed by *Task.getdviolbarvar*. (output)
- `dviolcone` (double) – Maximal violation of the dual solution associated with the dual conic constraints as computed by *Task.getdviolcones*. (output)

Groups *Solution information*

`Task.getsolutionslice`

```

void getsolutionslice
(soltype whichsol,
 solitem solitem,
 int first,
 int last,
 double[] 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` (*[soltype](#)*) – Selects a solution. (input)
- `solitem` (*[solitem](#)*) – Which part of the solution is required. (input)
- `first` (int) – First index in the sequence. (input)
- `last` (int) – Last index plus 1 in the sequence. (input)
- `values` (double[]) – The values in the required sequence are stored sequentially in `values`. (output)

**Groups** *[Solution - primal](#), [Solution - dual](#), [Solution information](#)*

`Task.getsparsesymmat`

```
void getsparsesymmat
(long idx,
 int[] subi,
 int[] subj,
 double[] valij)
```

Get a single symmetric matrix from the matrix store.

**Parameters**

- `idx` (long) – Index of the matrix to retrieve. (input)
- `subi` (int[]) – Row subscripts of the matrix non-zero elements. (output)
- `subj` (int[]) – Column subscripts of the matrix non-zero elements. (output)
- `valij` (double[]) – Coefficients of the matrix non-zero elements. (output)

**Groups** *[Problem data - semidefinite](#), [Inspecting the task](#)*

`Task.getstrparam`

```
string getstrparam
(sparam param,
 out int len)
```

```
void getstrparam
(sparam param,
 out int len,
 StringBuilder parvalue)
```

Obtains the value of a string parameter.

**Parameters**

- `param` (*[sparam](#)*) – Which parameter. (input)
- `len` (int) – The length of the parameter value. (output)
- `parvalue` (StringBuilder) – Parameter value. (output)

**Return** (string) – Parameter value.

**Groups** *[Names](#), [Parameters](#)*

`Task.getstrparamlen`

```
int getstrparamlen (sparam param)
```

```
void getstrparamlen
(sparam param,
 out int len)
```

Obtains the length of a string parameter.

**Parameters**

- `param` (*sparam*) – Which parameter. (input)
  - `len` (int) – The length of the parameter value. (output)
- Return** (int) – The length of the parameter value.
- Groups** *Names, Parameters*

Task.getsuc

```
void getsuc
(soltype whichsol,
 double[] suc)
```

Obtains the  $s_u^c$  vector for a solution.

**Parameters**

- `whichsol` (*soltype*) – Selects a solution. (input)
- `suc` (double[]) – Dual variables corresponding to the upper bounds on the constraints. (output)

**Groups** *Solution - dual*

Task.getsucslice

```
void getsucslice
(soltype whichsol,
 int first,
 int last,
 double[] suc)
```

Obtains a slice of the  $s_u^c$  vector for a solution.

**Parameters**

- `whichsol` (*soltype*) – Selects a solution. (input)
- `first` (int) – First index in the sequence. (input)
- `last` (int) – Last index plus 1 in the sequence. (input)
- `suc` (double[]) – Dual variables corresponding to the upper bounds on the constraints. (output)

**Groups** *Solution - dual*

Task.getsux

```
void getsux
(soltype whichsol,
 double[] sux)
```

Obtains the  $s_u^x$  vector for a solution.

**Parameters**

- `whichsol` (*soltype*) – Selects a solution. (input)
- `sux` (double[]) – Dual variables corresponding to the upper bounds on the variables. (output)

**Groups** *Solution - dual*

Task.getsuxslice

```
void getsuxslice
(soltype whichsol,
 int first,
 int last,
 double[] sux)
```

Obtains a slice of the  $s_u^x$  vector for a solution.

#### Parameters

- **whichsol** (*soltype*) – Selects a solution. (input)
- **first** (int) – First index in the sequence. (input)
- **last** (int) – Last index plus 1 in the sequence. (input)
- **sux** (double[]) – Dual variables corresponding to the upper bounds on the variables. (output)

**Groups** *Solution - dual*

**Task.getsymmatinfo**

```
void getsymmatinfo
(long idx,
 out int dim,
 out long nz,
 out symmattype type)
```

**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** (long) – Index of the matrix for which information is requested. (input)
- **dim** (int) – Returns the dimension of the requested matrix. (output)
- **nz** (long) – Returns the number of non-zeros in the requested matrix. (output)
- **type** (*symmattype*) – Returns the type of the requested matrix. (output)

**Groups** *Problem data - semidefinite, Inspecting the task*

**Task.gettaskname**

```
string gettaskname ()
```

```
void gettaskname (StringBuilder taskname)
```

Obtains the name assigned to the task.

**Parameters** **taskname** (StringBuilder) – Returns the task name. (output)

**Return** (string) – Returns the task name.

**Groups** *Names, Inspecting the task*

**Task.gettasknamelen**

```
int gettasknamelen ()
```

```
void gettasknamelen (out int len)
```

Obtains the length the task name.

**Parameters** **len** (int) – Returns the length of the task name. (output)

**Return** (int) – Returns the length of the task name.

## Groups *Names, Inspecting the task*

Task.getvarbound

```
void getvarbound
(int i,
 out boundkey bk,
 out double bl,
 out double bu)
```

Obtains bound information for one variable.

### Parameters

- **i** (**int**) – Index of the variable for which the bound information should be obtained. (input)
- **bk** (*boundkey*) – Bound keys. (output)
- **bl** (**double**) – Values for lower bounds. (output)
- **bu** (**double**) – Values for upper bounds. (output)

**Groups** *Problem data - linear part, Inspecting the task, Problem data - bounds, Problem data - variables*

Task.getvarboundslice

```
void getvarboundslice
(int first,
 int last,
 boundkey[] bk,
 double[] bl,
 double[] bu)
```

Obtains bounds information for a slice of the variables.

### Parameters

- **first** (**int**) – First index in the sequence. (input)
- **last** (**int**) – Last index plus 1 in the sequence. (input)
- **bk** (*boundkey* []) – Bound keys. (output)
- **bl** (**double**[]) – Values for lower bounds. (output)
- **bu** (**double**[]) – Values for upper bounds. (output)

**Groups** *Problem data - linear part, Inspecting the task, Problem data - bounds, Problem data - variables*

Task.getvarname

```
string getvarname (int j)
```

```
void getvarname
(int j,
 StringBuilder name)
```

Obtains the name of a variable.

### Parameters

- **j** (**int**) – Index of a variable. (input)
- **name** (**StringBuilder**) – Returns the required name. (output)

**Return** (**string**) – Returns the required name.

**Groups** *Names, Problem data - linear part, Problem data - variables, Inspecting the task*

Task.getvarnameindex

```
int getvarnameindex
(string somename,
 out int asgn)
```

```
void getvarnameindex
(string somename,
 out int asgn,
 out int index)
```

Checks whether the name **somename** has been assigned to any variable. If so, the index of the variable is reported.

**Parameters**

- **somename** (**string**) – The name which should be checked. (input)
- **asgn** (**int**) – Is non-zero if the name **somename** is assigned to a variable. (output)
- **index** (**int**) – If the name **somename** is assigned to a variable, then **index** is the index of the variable. (output)

**Return** (**int**) – 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

```
int getvarnamelen (int i)
```

```
void getvarnamelen
(int i,
 out int len)
```

Obtains the length of the name of a variable.

**Parameters**

- **i** (**int**) – Index of a variable. (input)
- **len** (**int**) – Returns the length of the indicated name. (output)

**Return** (**int**) – Returns the length of the indicated name.

**Groups** *Names, Problem data - linear part, Problem data - variables, Inspecting the task*

Task.getvartype

```
variabletype getvartype (int j)
```

```
void getvartype
(int j,
 out variabletype vartype)
```

Gets the variable type of one variable.

**Parameters**

- **j** (**int**) – Index of the variable. (input)

- **vartype** (*variabletype*) – Variable type of the  $j$ -th variable. (output)
- Return** (*variabletype*) – Variable type of the  $j$ -th variable.
- Groups** *Inspecting the task, Problem data - variables*

Task.getvartypelist

```
void getvartypelist
(int[] subj,
 variabletype[] vartype)
```

Obtains the variable type of one or more variables. Upon return **vartype[k]** is the variable type of variable **subj[k]**.

**Parameters**

- **subj** (**int[]**) – A list of variable indexes. (input)
- **vartype** (*variabletype*[]) – The variables types corresponding to the variables specified by **subj**. (output)

**Groups** *Inspecting the task, Problem data - variables*

Task.getxc

```
void getxc
(soltype whichsol,
 double[] xc)
```

Obtains the  $x^c$  vector for a solution.

**Parameters**

- **whichsol** (*soltype*) – Selects a solution. (input)
- **xc** (**double[]**) – Primal constraint solution. (output)

**Groups** *Solution - primal*

Task.getxcslice

```
void getxcslice
(soltype whichsol,
 int first,
 int last,
 double[] xc)
```

Obtains a slice of the  $x^c$  vector for a solution.

**Parameters**

- **whichsol** (*soltype*) – Selects a solution. (input)
- **first** (**int**) – First index in the sequence. (input)
- **last** (**int**) – Last index plus 1 in the sequence. (input)
- **xc** (**double[]**) – Primal constraint solution. (output)

**Groups** *Solution - primal*

Task.getxx

```
void getxx
(soltype whichsol,
 double[] xx)
```

Obtains the  $x^x$  vector for a solution.

#### Parameters

- `whichsol` (*[soltype](#)*) – Selects a solution. (input)
- `xx` (`double[]`) – Primal variable solution. (output)

Groups *[Solution](#) - [primal](#)*

`Task.getxxslice`

```
void getxxslice
(soltype whichsol,
 int first,
 int last,
 double[] xx)
```

Obtains a slice of the  $x^x$  vector for a solution.

#### Parameters

- `whichsol` (*[soltype](#)*) – Selects a solution. (input)
- `first` (`int`) – First index in the sequence. (input)
- `last` (`int`) – Last index plus 1 in the sequence. (input)
- `xx` (`double[]`) – Primal variable solution. (output)

Groups *[Solution](#) - [primal](#)*

`Task.gety`

```
void gety
(soltype whichsol,
 double[] y)
```

Obtains the  $y$  vector for a solution.

#### Parameters

- `whichsol` (*[soltype](#)*) – Selects a solution. (input)
- `y` (`double[]`) – Vector of dual variables corresponding to the constraints. (output)

Groups *[Solution](#) - [dual](#)*

`Task.getyslice`

```
void getyslice
(soltype whichsol,
 int first,
 int last,
 double[] y)
```

Obtains a slice of the  $y$  vector for a solution.

#### Parameters

- `whichsol` (*[soltype](#)*) – Selects a solution. (input)
- `first` (`int`) – First index in the sequence. (input)
- `last` (`int`) – Last index plus 1 in the sequence. (input)
- `y` (`double[]`) – Vector of dual variables corresponding to the constraints. (output)

Groups *[Solution](#) - [dual](#)*

`Task.initbasissolve`

```
void initbasissolve (int[] 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* (int[]) – The array of basis indexes to use. The array is interpreted as follows: If *basis*[*i*] ≤ *numcon* − 1, then  $x_{\text{basis}[i]}^c$  is in the basis at position *i*, otherwise  $x_{\text{basis}[i] - \text{numcon}}$  is in the basis at position *i*. (output)

**Groups** *Solving systems with basis matrix*

Task.inputdata

```
void inputdata
(int maxnumcon,
 int maxnumvar,
 double[] c,
 double cfix,
 int[] aptrb,
 int[] aptre,
 int[] asub,
 double[] aval,
 boundkey[] bkc,
 double[] blc,
 double[] buc,
 boundkey[] bkx,
 double[] blx,
 double[] bux)
```

```
void inputdata
(int maxnumcon,
 int maxnumvar,
 double[] c,
 double cfix,
 long[] aptrb,
 long[] aptre,
 int[] asub,
 double[] aval,
 boundkey[] bkc,
 double[] blc,
 double[] buc,
 boundkey[] bkx,
 double[] blx,
 double[] bux)
```

```
void inputdata
(int maxnumcon,
 int maxnumvar,
 int numcon,
 int numvar,
 double[] c,
```

(continues on next page)

```

double cfix,
long[] aptrb,
long[] aptre,
int[] asub,
double[] aval,
boundkey[] bkc,
double[] blc,
double[] buc,
boundkey[] bkc,
double[] blx,
double[] bux)

```

Input the linear part of an optimization task in one function call.

#### Parameters

- **maxnumcon** (int) – Number of preallocated constraints in the optimization task. (input)
- **maxnumvar** (int) – Number of preallocated variables in the optimization task. (input)
- **c** (double[]) – Linear terms of the objective as a dense vector. The length is the number of variables. (input)
- **cfix** (double) – Fixed term in the objective. (input)
- **aptrb** (int[]) – Row or column start pointers. (input)
- **aptrb** (long[]) – Row or column start pointers. (input)
- **aptre** (int[]) – Row or column end pointers. (input)
- **aptre** (long[]) – Row or column end pointers. (input)
- **asub** (int[]) – Coefficient subscripts. (input)
- **aval** (double[]) – Coefficient values. (input)
- **bkc** (*boundkey*[]) – Bound keys for the constraints. (input)
- **blc** (double[]) – Lower bounds for the constraints. (input)
- **buc** (double[]) – Upper bounds for the constraints. (input)
- **bkc** (*boundkey*[]) – Bound keys for the variables. (input)
- **blx** (double[]) – Lower bounds for the variables. (input)
- **bux** (double[]) – Upper bounds for the variables. (input)
- **numcon** (int) – Number of constraints. (input)
- **numvar** (int) – Number of variables. (input)

**Groups** *Problem data - linear part, Problem data - bounds, Problem data - constraints*

Task.isdoupname

```

void isdoupname
(string parname,
 out dparam param)

```

Checks whether **parname** is a valid double parameter name.

#### Parameters

- **parname** (string) – Parameter name. (input)
- **param** (*dparam*) – Returns the parameter corresponding to the name, if one exists. (output)

**Groups** *Parameters, Names*

Task.isintparname

```
void isintparname
(string parname,
 out iparam param)
```

Checks whether `parname` is a valid integer parameter name.

#### Parameters

- `parname` (*string*) – Parameter name. (input)
- `param` (*iparam*) – Returns the parameter corresponding to the name, if one exists. (output)

**Groups** *Parameters, Names*

`Task.isstrparname`

```
void isstrparname
(string parname,
 out sparam param)
```

Checks whether `parname` is a valid string parameter name.

#### Parameters

- `parname` (*string*) – Parameter name. (input)
- `param` (*sparam*) – Returns the parameter corresponding to the name, if one exists. (output)

**Groups** *Parameters, Names*

`Task.linkfiletostream`

```
void linkfiletostream
(streamtype whichstream,
 string filename,
 int append)
```

Directs all output from a task stream `whichstream` to a file `filename`.

#### Parameters

- `whichstream` (*streamtype*) – Index of the stream. (input)
- `filename` (*string*) – A valid file name. (input)
- `append` (*int*) – If this argument is 0 the output file will be overwritten, otherwise it will be appended to. (input)

**Groups** *Logging*

`Task.onesolutionsummary`

```
void onesolutionsummary
(streamtype whichstream,
 soltype whichsol)
```

Prints a short summary of a specified solution.

#### Parameters

- `whichstream` (*streamtype*) – Index of the stream. (input)
- `whichsol` (*soltype*) – Selects a solution. (input)

**Groups** *Logging, Solution information*

`Task.optimize`

```
rescode optimize ()
```

```
void optimize (out rescode 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*.

**Parameters** *trmcode* (*rescode*) – Is either *rescode.ok* or a termination response code. (output)

**Return** (*rescode*) – Is either *rescode.ok* or a termination response code.

**Groups** *Optimization*

Task.optimizermt

```
void optimizermt  
(string server,  
 string port,  
 out rescode trmcode)
```

Offload the optimization task to a solver server defined by *server:port*. The call will block until a result is available or the connection closes.

If the string parameter *sparam.remote\_access\_token* is not blank, it will be passed to the server as authentication.

**Parameters**

- *server* (string) – Name or IP address of the solver server. (input)
- *port* (string) – Network port of the solver server. (input)
- *trmcode* (*rescode*) – Is either *rescode.ok* or a termination response code. (output)

**Groups** *Remote optimization*

Task.optimizersummary

```
void optimizersummary (streamtype whichstream)
```

Prints a short summary with optimizer statistics from last optimization.

**Parameters** *whichstream* (*streamtype*) – Index of the stream. (input)

**Groups** *Logging*

Task.primalrepair

```
void primalrepair  
(double[] wlc,  
 double[] wuc,  
 double[] wlx,  
 double[] 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** (`double[]`) –  $(w_l^c)_i$  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 NULL, then all the weights are assumed to be 1. (input)
- **wuc** (`double[]`) –  $(w_u^c)_i$  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 NULL, then all the weights are assumed to be 1. (input)
- **wlx** (`double[]`) –  $(w_l^x)_j$  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 NULL, then all the weights are assumed to be 1. (input)
- **wux** (`double[]`) –  $(w_u^x)_i$  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 NULL, then all the weights are assumed to be 1. (input)

#### Groups *Infeasibility diagnostic*

Task.primalsensitivity

```
void primalsensitivity
(int[] subi,
 mark[] marki,
 int[] subj,
 mark[] markj,
 double[] leftpricei,
 double[] rightpricei,
 double[] leftrangei,
 double[] rightrangei,
 double[] leftpricej,
 double[] rightpricej,
 double[] leftrangej,
 double[] rightrangej)
```

```
void primalsensitivity
(int numi,
 int[] subi,
 mark[] marki,
 int numj,
 int[] subj,
 mark[] markj,
 double[] leftpricei,
 double[] rightpricei,
 double[] leftrangei,
 double[] rightrangei,
 double[] leftpricej,
 double[] rightpricej,
```

(continues on next page)



```
int nzj,
int[] subj,
double[] valj)
```

Change one column of the linear constraint matrix  $A$ . Resets all the elements in column  $j$  to zero and then sets

$$a_{\text{subj}[k],j} = \text{valj}[k], \quad k = 0, \dots, \text{nzj} - 1.$$

#### Parameters

- $j$  (int) – Index of a column in  $A$ . (input)
- $\text{subj}$  (int[]) – Row indexes of non-zero values in column  $j$  of  $A$ . (input)
- $\text{valj}$  (double[]) – New non-zero values of column  $j$  in  $A$ . (input)
- $\text{nzj}$  (int) – Number of non-zeros in column  $j$  of  $A$ . (input)

**Groups** *Problem data - linear part*

Task.putacollist

```
void putacollist
(int[] sub,
long[] ptrb,
long[] ptre,
int[] asub,
double[] aval)
```

```
void putacollist
(int num,
int[] sub,
long[] ptrb,
long[] ptre,
int[] asub,
double[] 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:

$$\text{for } i = 0, \dots, \text{num} - 1 \\ a_{\text{asub}[k], \text{sub}[i]} = \text{aval}[k], \quad k = \text{ptrb}[i], \dots, \text{ptre}[i] - 1.$$

#### Parameters

- $\text{sub}$  (int[]) – Indexes of columns that should be replaced, no duplicates. (input)
- $\text{ptrb}$  (long[]) – Array of pointers to the first element in each column. (input)
- $\text{ptre}$  (long[]) – Array of pointers to the last element plus one in each column. (input)
- $\text{asub}$  (int[]) – Row indexes of new elements. (input)
- $\text{aval}$  (double[]) – Coefficient values. (input)
- $\text{num}$  (int) – Number of columns of  $A$  to replace. (input)

**Groups** *Problem data - linear part*

Task.putaij

```
void putaij
(int i,
int j,
double aij)
```

Changes a coefficient in the linear coefficient matrix  $A$  using the method

$$a_{i,j} = \text{aij}.$$

**Parameters**

- **i** (int) – Constraint (row) index. (input)
- **j** (int) – Variable (column) index. (input)
- **aij** (double) – New coefficient for  $a_{i,j}$ . (input)

**Groups** *Problem data - linear part*

Task.putaijlist

```
void putaijlist
(int[] subi,
 int[] subj,
 double[] valij)
```

```
void putaijlist
(long num,
 int[] subi,
 int[] subj,
 double[] valij)
```

Changes one or more coefficients in  $A$  using the method

$$a_{\text{subi}[k], \text{subj}[k]} = \text{valij}[k], \quad k = 0, \dots, \text{num} - 1.$$

Duplicates are not allowed.

**Parameters**

- **subi** (int[]) – Constraint (row) indices. (input)
- **subj** (int[]) – Variable (column) indices. (input)
- **valij** (double[]) – New coefficient values for  $a_{i,j}$ . (input)
- **num** (long) – Number of coefficients that should be changed. (input)

**Groups** *Problem data - linear part*

Task.putarow

```
void putarow
(int i,
 int[] subi,
 double[] vali)
```

```
void putarow
(int i,
 int nzi,
 int[] subi,
 double[] vali)
```

Change one row of the linear constraint matrix  $A$ . Resets all the elements in row  $i$  to zero and then sets

$$a_{i, \text{subi}[k]} = \text{vali}[k], \quad k = 0, \dots, \text{nzi} - 1.$$

**Parameters**

- **i** (int) – Index of a row in  $A$ . (input)

- `subi (int[])` – Column indexes of non-zero values in row  $i$  of  $A$ . (input)
- `vali (double[])` – New non-zero values of row  $i$  in  $A$ . (input)
- `nzi (int)` – Number of non-zeros in row  $i$  of  $A$ . (input)

**Groups** *Problem data - linear part*

`Task.putarowlist`

```
void putarowlist
(int[] sub,
 long[] ptrb,
 long[] ptre,
 int[] asub,
 double[] aval)
```

```
void putarowlist
(int num,
 int[] sub,
 long[] ptrb,
 long[] ptre,
 int[] asub,
 double[] 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:

$$\text{for } i = 0, \dots, num - 1 \\ a_{\text{sub}[i], \text{asub}[k]} = \text{aval}[k], \quad k = \text{ptrb}[i], \dots, \text{ptre}[i] - 1.$$

**Parameters**

- `sub (int[])` – Indexes of rows that should be replaced, no duplicates. (input)
- `ptrb (long[])` – Array of pointers to the first element in each row. (input)
- `ptre (long[])` – Array of pointers to the last element plus one in each row. (input)
- `asub (int[])` – Column indexes of new elements. (input)
- `aval (double[])` – Coefficient values. (input)
- `num (int)` – Number of rows of  $A$  to replace. (input)

**Groups** *Problem data - linear part*

`Task.putatruncatetol`

```
void putatruncatetol (double tolzero)
```

Truncates (sets to zero) all elements in  $A$  that satisfy

$$|a_{i,j}| \leq \text{tolzero}.$$

**Parameters** `tolzero (double)` – Truncation tolerance. (input)

**Groups** *Problem data - linear part*

`Task.putbarablocktriplet`

```
void putbarablocktriplet
(long num,
 int[] subi,
 int[] subj,
 int[] subk,
 int[] subl,
 double[] valijkl)
```

Inputs the  $\bar{A}$  matrix in block triplet form.

#### Parameters

- num (long) – Number of elements in the block triplet form. (input)
- subi (int[]) – Constraint index. (input)
- subj (int[]) – Symmetric matrix variable index. (input)
- subk (int[]) – Block row index. (input)
- subl (int[]) – Block column index. (input)
- valijkl (double[]) – The numerical value associated with each block triplet. (input)

Groups *Problem data - semidefinite*

Task.putbaraij

```
void putbaraij
(int i,
 int j,
 long[] sub,
 double[] weights)
```

This function sets one element in the  $\bar{A}$  matrix.

Each element in the  $\bar{A}$  matrix is a weighted sum of symmetric matrices from the symmetric matrix storage  $E$ , so  $\bar{A}_{ij}$  is a symmetric matrix. By default all elements in  $\bar{A}$  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 (int) – Row index of  $\bar{A}$ . (input)
- j (int) – Column index of  $\bar{A}$ . (input)
- sub (long[]) – Indices in  $E$  of the matrices appearing in the weighted sum for  $\bar{A}_{ij}$ . (input)
- weights (double[]) –  $\text{weights}[k]$  is the coefficient of the  $\text{sub}[k]$ -th element of  $E$  in the weighted sum forming  $\bar{A}_{ij}$ . (input)

Groups *Problem data - semidefinite*

Task.putbaraijlist

```
void putbaraijlist
(int[] subi,
 int[] subj,
 long[] alphaptrb,
 long[] alphaptre,
 long[] matidx,
 double[] weights)
```

This function sets a list of elements in the  $\bar{A}$  matrix.

Each element in the  $\bar{A}$  matrix is a weighted sum of symmetric matrices from the symmetric matrix storage  $E$ , so  $\bar{A}_{ij}$  is a symmetric matrix. By default all elements in  $\bar{A}$  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 (int[]) – Row index of  $\bar{A}$ . (input)

- `subj` (`int[]`) – Column index of  $\bar{A}$ . (input)
- `alphaptrb` (`long[]`) – Start entries for terms in the weighted sum that forms  $\bar{A}_{ij}$ . (input)
- `alphaptre` (`long[]`) – End entries for terms in the weighted sum that forms  $\bar{A}_{ij}$ . (input)
- `matidx` (`long[]`) – Indices in  $E$  of the matrices appearing in the weighted sum for  $\bar{A}_{ij}$ . (input)
- `weights` (`double[]`) – `weights[k]` is the coefficient of the `sub[k]`-th element of  $E$  in the weighted sum forming  $\bar{A}_{ij}$ . (input)

**Groups** *Problem data - semidefnite*

`Task.putbararowlist`

```
void putbararowlist
(int[] subi,
 long[] ptrb,
 long[] ptre,
 int[] subj,
 long[] nummat,
 long[] matidx,
 double[] weights)
```

This function replaces a list of rows in the  $\bar{A}$  matrix.

**Parameters**

- `subi` (`int[]`) – Row indexes of  $\bar{A}$ . (input)
- `ptrb` (`long[]`) – Start of rows in  $\bar{A}$ . (input)
- `ptre` (`long[]`) – End of rows in  $\bar{A}$ . (input)
- `subj` (`int[]`) – Column index of  $\bar{A}$ . (input)
- `nummat` (`long[]`) – Number of entries in weighted sum of matrixes. (input)
- `matidx` (`long[]`) – Matrix indexes for weighted sum of matrixes. (input)
- `weights` (`double[]`) – Weights for weighted sum of matrixes. (input)

**Groups** *Problem data - semidefnite*

`Task.putbarcblocktriplet`

```
void putbarcblocktriplet
(long num,
 int[] subj,
 int[] subk,
 int[] subl,
 double[] valjkl)
```

Inputs the  $\bar{C}$  matrix in block triplet form.

**Parameters**

- `num` (`long`) – Number of elements in the block triplet form. (input)
- `subj` (`int[]`) – Symmetric matrix variable index. (input)
- `subk` (`int[]`) – Block row index. (input)
- `subl` (`int[]`) – Block column index. (input)
- `valjkl` (`double[]`) – The numerical value associated with each block triplet. (input)

**Groups** *Problem data - semidefnite*

`Task.putbarcj`



### Parameters

- `whichsol` (*soltype*) – Selects a solution. (input)
- `j` (int) – Index of the semidefinite variable. (input)
- `barxj` (double[]) – Value of  $\bar{X}_j$ . Format as in *Task.getbarxj*. (input)

**Groups** *Solution - semidefinite*

`Task.putcfix`

```
void putcfix (double cfix)
```

Replaces the fixed term in the objective by a new one.

**Parameters** `cfix` (double) – Fixed term in the objective. (input)

**Groups** *Problem data - linear part, Problem data - objective*

`Task.putcj`

```
void putcj  
(int j,  
 double cj)
```

Modifies one coefficient in the linear objective vector  $c$ , i.e.

$$c_j = cj.$$

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` (int) – Index of the variable for which  $c$  should be changed. (input)
- `cj` (double) – New value of  $c_j$ . (input)

**Groups** *Problem data - linear part, Problem data - objective*

`Task.putclist`

```
void putclist  
(int[] subj,  
 double[] val)
```

```
void putclist  
(int num,  
 int[] subj,  
 double[] val)
```

Modifies the coefficients in the linear term  $c$  in the objective using the principle

$$c_{\text{subj}[t]} = \text{val}[t], \quad t = 0, \dots, \text{num} - 1.$$

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` (int[]) – Indices of variables for which the coefficient in  $c$  should be changed. (input)
- `val` (double[]) – New numerical values for coefficients in  $c$  that should be modified. (input)

- `num (int)` – Number of coefficients that should be changed. (input)

**Groups** *Problem data - linear part, Problem data - variables, Problem data - objective*

`Task.putconbound`

```
void putconbound
(int i,
 boundkey bkc,
 double blc,
 double 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 (int)` – Index of the constraint. (input)
- `bkc (boundkey)` – New bound key. (input)
- `blc (double)` – New lower bound. (input)
- `buc (double)` – New upper bound. (input)

**Groups** *Problem data - linear part, Problem data - constraints, Problem data - bounds*

`Task.putconboundlist`

```
void putconboundlist
(int[] sub,
 boundkey[] bkc,
 double[] blc,
 double[] buc)
```

```
void putconboundlist
(int num,
 int[] sub,
 boundkey[] bkc,
 double[] blc,
 double[] 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 (int[])` – List of constraint indexes. (input)
- `bkc (boundkey[])` – Bound keys for the constraints. (input)
- `blc (double[])` – Lower bounds for the constraints. (input)
- `buc (double[])` – Upper bounds for the constraints. (input)
- `num (int)` – Number of bounds that should be changed. (input)

**Groups** *Problem data - linear part, Problem data - constraints, Problem data - bounds*

`Task.putconboundlistconst`

```
void putconboundlistconst
(int[] sub,
 boundkey bkc,
 double blc,
 double buc)
```

```
void putconboundlistconst
(int num,
 int[] sub,
 boundkey bkc,
 double blc,
 double buc)
```

Changes the bounds for one or more constraints. Data checks are performed as in *Task.putconbound*.

#### Parameters

- sub (int[]) – List of constraint indexes. (input)
- bkc (*boundkey*) – New bound key for all constraints in the list. (input)
- blc (double) – New lower bound for all constraints in the list. (input)
- buc (double) – New upper bound for all constraints in the list. (input)
- num (int) – Number of bounds that should be changed. (input)

**Groups** *Problem data - linear part, Problem data - constraints, Problem data - bounds*

Task.putconboundslice

```
void putconboundslice
(int first,
 int last,
 boundkey[] bkc,
 double[] blc,
 double[] buc)
```

Changes the bounds for a slice of the constraints. Data checks are performed as in *Task.putconbound*.

#### Parameters

- first (int) – First index in the sequence. (input)
- last (int) – Last index plus 1 in the sequence. (input)
- bkc (*boundkey*[]) – Bound keys for the constraints. (input)
- blc (double[]) – Lower bounds for the constraints. (input)
- buc (double[]) – Upper bounds for the constraints. (input)

**Groups** *Problem data - linear part, Problem data - constraints, Problem data - bounds*

Task.putconboundsliceconst

```
void putconboundsliceconst
(int first,
 int last,
 boundkey bkc,
 double blc,
 double buc)
```

Changes the bounds for a slice of the constraints. Data checks are performed as in *Task.putconbound*.

#### Parameters

- first (int) – First index in the sequence. (input)
- last (int) – Last index plus 1 in the sequence. (input)
- bkc (*boundkey*) – New bound key for all constraints in the slice. (input)
- blc (double) – New lower bound for all constraints in the slice. (input)
- buc (double) – New upper bound for all constraints in the slice. (input)

**Groups** *Problem data - linear part, Problem data - constraints, Problem data - bounds*

Task.putcone

```
void putcone
(int k,
 conetype ct,
 double coneapar,
 int[] submem)
```

```
void putcone
(int k,
 conetype ct,
 double coneapar,
 int nummem,
 int[] submem)
```

Replaces a conic constraint.

**Parameters**

- **k** (**int**) – Index of the cone. (input)
- **ct** (*conetype*) – Specifies the type of the cone. (input)
- **coneapar** (**double**) – For the power cone it denotes the exponent alpha. For other cone types it is unused and can be set to 0. (input)
- **submem** (**int**[]) – Variable subscripts of the members in the cone. (input)
- **nummem** (**int**) – Number of member variables in the cone. (input)

**Groups** *Problem data - cones*

Task.putconename

```
void putconename
(int j,
 string name)
```

Sets the name of a cone.

**Parameters**

- **j** (**int**) – Index of the cone. (input)
- **name** (**string**) – The name of the cone. (input)

**Groups** *Names, Problem data - cones*

Task.putconname

```
void putconname
(int i,
 string name)
```

Sets the name of a constraint.

**Parameters**

- **i** (**int**) – Index of the constraint. (input)
- **name** (**string**) – The name of the constraint. (input)

**Groups** *Names, Problem data - constraints, Problem data - linear part*

Task.putconsolutioni

```

void putconsolutioni
(int i,
 soltype whichsol,
 stakey sk,
 double x,
 double sl,
 double su)

```

Sets the primal and dual solution information for a single constraint.

#### Parameters

- **i** (**int**) – Index of the constraint. (input)
- **whichsol** (***soltype***) – Selects a solution. (input)
- **sk** (***stakey***) – Status key of the constraint. (input)
- **x** (**double**) – Primal solution value of the constraint. (input)
- **sl** (**double**) – Solution value of the dual variable associated with the lower bound. (input)
- **su** (**double**) – Solution value of the dual variable associated with the upper bound. (input)

**Groups** *Solution information, Solution - primal, Solution - dual*

**Task.putcslice**

```

void putcslice
(int first,
 int last,
 double[] slice)

```

Modifies a slice in the linear term  $c$  in the objective using the principle

$$c_j = \text{slice}[j - \text{first}], \quad j = \text{first}, \dots, \text{last} - 1$$

Data checks are performed as in *Task.putcj*.

#### Parameters

- **first** (**int**) – First element in the slice of  $c$ . (input)
- **last** (**int**) – Last element plus 1 of the slice in  $c$  to be changed. (input)
- **slice** (**double[]**) – New numerical values for coefficients in  $c$  that should be modified. (input)

**Groups** *Problem data - linear part, Problem data - objective*

**Task.putdouparam**

```

void putdouparam
(dparam param,
 double parvalue)

```

Sets the value of a double parameter.

#### Parameters

- **param** (***dparam***) – Which parameter. (input)
- **parvalue** (**double**) – Parameter value. (input)

**Groups** *Parameters*

**Task.putintparam**

```
void putintparam
(iparam param,
 int parvalue)
```

Sets the value of an integer parameter.

**Parameters**

- **param** (*iparam*) – Which parameter. (input)
- **parvalue** (*int*) – Parameter value. (input)

**Groups** *Parameters*

**Task.**putmaxnumanz

```
void putmaxnumanz (long 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** (*long*) – Number of preallocated non-zeros in  $A$ . (input)

**Groups** *Environment and task management, Problem data - semidefinite*

**Task.**putmaxnumbarvar

```
void putmaxnumbarvar (int 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** (*int*) – Number of preallocated symmetric matrix variables. (input)

**Groups** *Environment and task management, Problem data - semidefinite*

**Task.**putmaxnumcon

```
void putmaxnumcon (int 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** (*int*) – Number of preallocated constraints in the optimization task. (input)

**Groups** *Environment and task management, Problem data - constraints*

`Task.putmaxnumcone`

```
void putmaxnumcone (int maxnumcone)
```

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 (int)` – Number of preallocated conic constraints in the optimization task. (input)

**Groups** *Environment and task management, Problem data - cones*

`Task.putmaxnumqnz`

```
void putmaxnumqnz (long 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 (long)` – Number of non-zero elements preallocated in quadratic coefficient matrices. (input)

**Groups** *Environment and task management, Problem data - quadratic part*

`Task.putmaxnumvar`

```
void putmaxnumvar (int 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 (int)` – Number of preallocated variables in the optimization task. (input)

**Groups** *Environment and task management, Problem data - variables*

`Task.putnadoupam`

```
void putnadoupam  
  (string paramname,  
   double parvalue)
```

Sets the value of a named double parameter.

**Parameters**

- `paramname` (`string`) – Name of a parameter. (input)
- `parvalue` (`double`) – Parameter value. (input)

**Groups** *Parameters*

`Task.putnaintparam`

```
void putnaintparam
(string paramname,
 int parvalue)
```

Sets the value of a named integer parameter.

**Parameters**

- `paramname` (`string`) – Name of a parameter. (input)
- `parvalue` (`int`) – Parameter value. (input)

**Groups** *Parameters*

`Task.putnastrparam`

```
void putnastrparam
(string paramname,
 string parvalue)
```

Sets the value of a named string parameter.

**Parameters**

- `paramname` (`string`) – Name of a parameter. (input)
- `parvalue` (`string`) – Parameter value. (input)

**Groups** *Parameters*

`Task.putobjname`

```
void putobjname (string objname)
```

Assigns a new name to the objective.

**Parameters** `objname` (`string`) – Name of the objective. (input)

**Groups** *Problem data - linear part, Names, Problem data - objective*

`Task.putobjsense`

```
void putobjsense (objsense sense)
```

Sets the objective sense of the task.

**Parameters** `sense` (*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`

```
void putoptserverhost (string host)
```

Specify an OptServer URL for remote calls. The URL should contain protocol, host and port in the form `http://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 NULL deactivates this redirection.

**Parameters** `host` (**string**) – A URL specifying the optimization server to be used.  
(input)

**Groups** *Remote optimization*

`Task.putparam`

```
void putparam
(string parname,
 string parvalue)
```

Checks if `parname` is valid parameter name. If it is, the parameter is assigned the value specified by `parvalue`.

**Parameters**

- `parname` (**string**) – Parameter name. (input)
- `parvalue` (**string**) – Parameter value. (input)

**Groups** *Parameters*

`Task.putqcon`

```
void putqcon
(int[] qcsbk,
 int[] qcsubi,
 int[] qcsubj,
 double[] qcval)
```

```
void putqcon
(int numqcnz,
 int[] qcsbk,
 int[] qcsubi,
 int[] qcsubj,
 double[] qcval)
```

Replace all quadratic entries in the constraints. The list of constraints has the form

$$l_k^c \leq \frac{1}{2} \sum_{i=0}^{numvar-1} \sum_{j=0}^{numvar-1} q_{ij}^k x_i x_j + \sum_{j=0}^{numvar-1} a_{kj} x_j \leq u_k^c, \quad k = 0, \dots, m-1.$$

This function sets all the quadratic terms to zero and then performs the update:

$$q_{qcsubi[t], qcsubj[t]}^{qcsubk[t]} = q_{qcsubj[t], qcsubi[t]}^{qcsubk[t]} = q_{qcsubj[t], qcsubi[t]}^{qcsubk[t]} + qcval[t],$$

for  $t = 0, \dots, numqcnz - 1$ .

Please note that:

- For large problems it is essential for the efficiency that the function *Task.putmaxnumqcnz* 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

- `qcsbck (int [])` – Constraint subscripts for quadratic coefficients. (input)
- `qcsubi (int [])` – Row subscripts for quadratic constraint matrix. (input)
- `qcsubj (int [])` – Column subscripts for quadratic constraint matrix. (input)
- `qcval (double [])` – Quadratic constraint coefficient values. (input)
- `numqcnz (int)` – Number of quadratic terms. (input)

Groups *Problem data - quadratic part*

`Task.putqconk`

```
void putqconk
(int k,
 int[] qcsubi,
 int[] qcsubj,
 double[] qcval)
```

```
void putqconk
(int k,
 int numqcnz,
 int[] qcsubi,
 int[] qcsubj,
 double[] 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 (int)` – The constraint in which the new  $Q$  elements are inserted. (input)
- `qcsubi (int [])` – Row subscripts for quadratic constraint matrix. (input)
- `qcsubj (int [])` – Column subscripts for quadratic constraint matrix. (input)
- `qcval (double [])` – Quadratic constraint coefficient values. (input)
- `numqcnz (int)` – Number of quadratic terms. (input)

Groups *Problem data - quadratic part*

`Task.putqobj`

```
void putqobj
(int[] qosubi,
 int[] qosubj,
 double[] qoval)
```

```
void putqobj
(int numqonz,
 int[] qosubi,
 int[] qosubj,
 double[] qoval)
```



```
void putskcslice
(soltype whichsol,
 int first,
 int last,
 stakey[] skc)
```

Sets the status keys for a slice of the constraints.

#### Parameters

- `whichsol` (*soltype*) – Selects a solution. (input)
- `first` (int) – First index in the sequence. (input)
- `last` (int) – Last index plus 1 in the sequence. (input)
- `skc` (*stakey*[]) – Status keys for the constraints. (input)

**Groups** *Solution information*

`Task.putskx`

```
void putskx
(soltype whichsol,
 stakey[] skx)
```

Sets the status keys for the scalar variables.

#### Parameters

- `whichsol` (*soltype*) – Selects a solution. (input)
- `skx` (*stakey*[]) – Status keys for the variables. (input)

**Groups** *Solution information*

`Task.putskxslice`

```
void putskxslice
(soltype whichsol,
 int first,
 int last,
 stakey[] skx)
```

Sets the status keys for a slice of the variables.

#### Parameters

- `whichsol` (*soltype*) – Selects a solution. (input)
- `first` (int) – First index in the sequence. (input)
- `last` (int) – Last index plus 1 in the sequence. (input)
- `skx` (*stakey*[]) – Status keys for the variables. (input)

**Groups** *Solution information*

`Task.putslc`

```
void putslc
(soltype whichsol,
 double[] slc)
```

Sets the  $s_l^c$  vector for a solution.

#### Parameters

- `whichsol` (*soltype*) – Selects a solution. (input)

- `slc (double[])` – Dual variables corresponding to the lower bounds on the constraints. (input)

**Groups** *Solution - dual*

`Task.putslcslice`

```
void putslcslice
(soltype whichsol,
 int first,
 int last,
 double[] slc)
```

Sets a slice of the  $s_l^c$  vector for a solution.

**Parameters**

- `whichsol (soltype)` – Selects a solution. (input)
- `first (int)` – First index in the sequence. (input)
- `last (int)` – Last index plus 1 in the sequence. (input)
- `slc (double[])` – Dual variables corresponding to the lower bounds on the constraints. (input)

**Groups** *Solution - dual*

`Task.putslx`

```
void putslx
(soltype whichsol,
 double[] slx)
```

Sets the  $s_l^x$  vector for a solution.

**Parameters**

- `whichsol (soltype)` – Selects a solution. (input)
- `slx (double[])` – Dual variables corresponding to the lower bounds on the variables. (input)

**Groups** *Solution - dual*

`Task.putslxslice`

```
void putslxslice
(soltype whichsol,
 int first,
 int last,
 double[] slx)
```

Sets a slice of the  $s_l^x$  vector for a solution.

**Parameters**

- `whichsol (soltype)` – Selects a solution. (input)
- `first (int)` – First index in the sequence. (input)
- `last (int)` – Last index plus 1 in the sequence. (input)
- `slx (double[])` – Dual variables corresponding to the lower bounds on the variables. (input)

**Groups** *Solution - dual*

`Task.putsnx`

```
void putsnx
(soltype whichsol,
 double[] sux)
```

Sets the  $s_n^x$  vector for a solution.

#### Parameters

- `whichsol` (*soltype*) – Selects a solution. (input)
- `sux` (`double[]`) – Dual variables corresponding to the upper bounds on the variables. (input)

**Groups** *Solution - dual*

`Task.putsnxslice`

```
void putsnxslice
(soltype whichsol,
 int first,
 int last,
 double[] snx)
```

Sets a slice of the  $s_n^x$  vector for a solution.

#### Parameters

- `whichsol` (*soltype*) – Selects a solution. (input)
- `first` (`int`) – First index in the sequence. (input)
- `last` (`int`) – Last index plus 1 in the sequence. (input)
- `snx` (`double[]`) – Dual variables corresponding to the conic constraints on the variables. (input)

**Groups** *Solution - dual*

`Task.putsolution`

```
void putsolution
(soltype whichsol,
 stakey[] skc,
 stakey[] skx,
 stakey[] skn,
 double[] xc,
 double[] xx,
 double[] y,
 double[] slc,
 double[] suc,
 double[] slx,
 double[] sux,
 double[] snx)
```

Inserts a solution into the task.

#### Parameters

- `whichsol` (*soltype*) – Selects a solution. (input)
- `skc` (*stakey*[]) – Status keys for the constraints. (input)
- `skx` (*stakey*[]) – Status keys for the variables. (input)
- `skn` (*stakey*[]) – Status keys for the conic constraints. (input)
- `xc` (`double[]`) – Primal constraint solution. (input)
- `xx` (`double[]`) – Primal variable solution. (input)
- `y` (`double[]`) – Vector of dual variables corresponding to the constraints. (input)

- `slc (double[])` – Dual variables corresponding to the lower bounds on the constraints. (input)
- `suc (double[])` – Dual variables corresponding to the upper bounds on the constraints. (input)
- `slx (double[])` – Dual variables corresponding to the lower bounds on the variables. (input)
- `sux (double[])` – Dual variables corresponding to the upper bounds on the variables. (input)
- `snx (double[])` – Dual variables corresponding to the conic constraints on the variables. (input)

**Groups** *Solution information, Solution - primal, Solution - dual*

`Task.putsolutionyi`

```
void putsolutionyi
(int i,
 soltype whichsol,
 double y)
```

Inputs the dual variable of a solution.

**Parameters**

- `i (int)` – Index of the dual variable. (input)
- `whichsol (soltype)` – Selects a solution. (input)
- `y (double)` – Solution value of the dual variable. (input)

**Groups** *Solution information, Solution - dual*

`Task.putstrparam`

```
void putstrparam
(sparam param,
 string parvalue)
```

Sets the value of a string parameter.

**Parameters**

- `param (sparam)` – Which parameter. (input)
- `parvalue (string)` – Parameter value. (input)

**Groups** *Parameters*

`Task.putsuc`

```
void putsuc
(soltype whichsol,
 double[] suc)
```

Sets the  $s_u^c$  vector for a solution.

**Parameters**

- `whichsol (soltype)` – Selects a solution. (input)
- `suc (double[])` – Dual variables corresponding to the upper bounds on the constraints. (input)

**Groups** *Solution - dual*

`Task.putsucslice`

```
void putsucslice
(soltype whichsol,
 int first,
 int last,
 double[] suc)
```

Sets a slice of the  $s_u^c$  vector for a solution.

**Parameters**

- **whichsol** (*soltype*) – Selects a solution. (input)
- **first** (int) – First index in the sequence. (input)
- **last** (int) – Last index plus 1 in the sequence. (input)
- **suc** (double[]) – Dual variables corresponding to the upper bounds on the constraints. (input)

**Groups** *Solution - dual*

Task.putsux

```
void putsux
(soltype whichsol,
 double[] sux)
```

Sets the  $s_u^x$  vector for a solution.

**Parameters**

- **whichsol** (*soltype*) – Selects a solution. (input)
- **sux** (double[]) – Dual variables corresponding to the upper bounds on the variables. (input)

**Groups** *Solution - dual*

Task.putsuxslice

```
void putsuxslice
(soltype whichsol,
 int first,
 int last,
 double[] sux)
```

Sets a slice of the  $s_u^x$  vector for a solution.

**Parameters**

- **whichsol** (*soltype*) – Selects a solution. (input)
- **first** (int) – First index in the sequence. (input)
- **last** (int) – Last index plus 1 in the sequence. (input)
- **sux** (double[]) – Dual variables corresponding to the upper bounds on the variables. (input)

**Groups** *Solution - dual*

Task.puttaskname

```
void puttaskname (string taskname)
```

Assigns a new name to the task.

**Parameters** **taskname** (string) – Name assigned to the task. (input)

**Groups** *Names, Environment and task management*

## Task.putvarbound

```
void putvarbound
(int j,
 boundkey bxx,
 double blx,
 double 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 (int) – Index of the variable. (input)
- bxx (*boundkey*) – New bound key. (input)
- blx (double) – New lower bound. (input)
- bux (double) – New upper bound. (input)

**Groups** *Problem data - linear part, Problem data - variables, Problem data - bounds*

## Task.putvarboundlist

```
void putvarboundlist
(int[] sub,
 boundkey[] bxx,
 double[] blx,
 double[] bux)
```

```
void putvarboundlist
(int num,
 int[] sub,
 boundkey[] bxx,
 double[] blx,
 double[] 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 (int[]) – List of variable indexes. (input)
- bxx (*boundkey*[]) – Bound keys for the variables. (input)
- blx (double[]) – Lower bounds for the variables. (input)
- bux (double[]) – Upper bounds for the variables. (input)
- num (int) – Number of bounds that should be changed. (input)

**Groups** *Problem data - linear part, Problem data - variables, Problem data - bounds*

## Task.putvarboundlistconst

```
void putvarboundlistconst
(int[] sub,
 boundkey bxx,
 double blx,
 double bux)
```

```
void putvarboundlistconst
(int num,
 int[] sub,
 boundkey bxx,
 double blx,
 double bux)
```

Changes the bounds for one or more variables. Data checks are performed as in *Task.putvarbound*.

#### Parameters

- sub (int[]) – List of variable indexes. (input)
- bxx (*boundkey*) – New bound key for all variables in the list. (input)
- blx (double) – New lower bound for all variables in the list. (input)
- bux (double) – New upper bound for all variables in the list. (input)
- num (int) – Number of bounds that should be changed. (input)

**Groups** *Problem data - linear part, Problem data - variables, Problem data - bounds*

Task.putvarboundslice

```
void putvarboundslice
(int first,
 int last,
 boundkey[] bxx,
 double[] blx,
 double[] bux)
```

Changes the bounds for a slice of the variables. Data checks are performed as in *Task.putvarbound*.

#### Parameters

- first (int) – First index in the sequence. (input)
- last (int) – Last index plus 1 in the sequence. (input)
- bxx (*boundkey*[]) – Bound keys for the variables. (input)
- blx (double[]) – Lower bounds for the variables. (input)
- bux (double[]) – Upper bounds for the variables. (input)

**Groups** *Problem data - linear part, Problem data - variables, Problem data - bounds*

Task.putvarboundsliceconst

```
void putvarboundsliceconst
(int first,
 int last,
 boundkey bxx,
 double blx,
 double bux)
```

Changes the bounds for a slice of the variables. Data checks are performed as in *Task.putvarbound*.

#### Parameters

- first (int) – First index in the sequence. (input)
- last (int) – Last index plus 1 in the sequence. (input)
- bxx (*boundkey*) – New bound key for all variables in the slice. (input)
- blx (double) – New lower bound for all variables in the slice. (input)
- bux (double) – New upper bound for all variables in the slice. (input)

**Groups** *Problem data - linear part, Problem data - variables, Problem data - bounds*

Task.putvarname

```
void putvarname
(int j,
 string name)
```

Sets the name of a variable.

**Parameters**

- **j** (**int**) – Index of the variable. (input)
- **name** (**string**) – The variable name. (input)

**Groups** *Names, Problem data - variables, Problem data - linear part*

Task.putvarsolutionj

```
void putvarsolutionj
(int j,
 soltype whichsol,
 stakekey sk,
 double x,
 double sl,
 double su,
 double sn)
```

Sets the primal and dual solution information for a single variable.

**Parameters**

- **j** (**int**) – Index of the variable. (input)
- **whichsol** (***soltype***) – Selects a solution. (input)
- **sk** (***stakekey***) – Status key of the variable. (input)
- **x** (**double**) – Primal solution value of the variable. (input)
- **sl** (**double**) – Solution value of the dual variable associated with the lower bound. (input)
- **su** (**double**) – Solution value of the dual variable associated with the upper bound. (input)
- **sn** (**double**) – Solution value of the dual variable associated with the conic constraint. (input)

**Groups** *Solution information, Solution - primal, Solution - dual*

Task.putvartype

```
void putvartype
(int j,
 variabletype vartype)
```

Sets the variable type of one variable.

**Parameters**

- **j** (**int**) – Index of the variable. (input)
- **vartype** (***variabletype***) – The new variable type. (input)

**Groups** *Problem data - variables*

Task.putvartypelist

```
void putvartypelist
(int[] subj,
 variabletype[] vartype)
```

```
void putvartypelist
(int num,
 int[] subj,
 variabletype[] 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` (`int[]`) – A list of variable indexes for which the variable type should be changed. (input)
- `vartype` (*`variabletype`* `[]`) – A list of variable types that should be assigned to the variables specified by `subj`. (input)
- `num` (`int`) – Number of variables for which the variable type should be set. (input)

**Groups** *Problem data - variables*

`Task.putxc`

```
void putxc
(soltype whichsol,
 double[] xc)
```

Sets the  $x^c$  vector for a solution.

#### Parameters

- `whichsol` (*`soltype`*) – Selects a solution. (input)
- `xc` (`double[]`) – Primal constraint solution. (output)

**Groups** *Solution - primal*

`Task.putxcslice`

```
void putxcslice
(soltype whichsol,
 int first,
 int last,
 double[] xc)
```

Sets a slice of the  $x^c$  vector for a solution.

#### Parameters

- `whichsol` (*`soltype`*) – Selects a solution. (input)
- `first` (`int`) – First index in the sequence. (input)
- `last` (`int`) – Last index plus 1 in the sequence. (input)
- `xc` (`double[]`) – Primal constraint solution. (input)

**Groups** *Solution - primal*

`Task.putxx`

```
void putxx
(soltype whichsol,
 double[] xx)
```

Sets the  $x^x$  vector for a solution.

**Parameters**

- `whichsol` (*[soltype](#)*) – Selects a solution. (input)
- `xx` (`double[]`) – Primal variable solution. (input)

**Groups** *[Solution](#) - [primal](#)*

`Task.putxxslice`

```
void putxxslice
(soltype whichsol,
 int first,
 int last,
 double[] xx)
```

Sets a slice of the  $x^x$  vector for a solution.

**Parameters**

- `whichsol` (*[soltype](#)*) – Selects a solution. (input)
- `first` (`int`) – First index in the sequence. (input)
- `last` (`int`) – Last index plus 1 in the sequence. (input)
- `xx` (`double[]`) – Primal variable solution. (input)

**Groups** *[Solution](#) - [primal](#)*

`Task.puty`

```
void puty
(soltype whichsol,
 double[] y)
```

Sets the  $y$  vector for a solution.

**Parameters**

- `whichsol` (*[soltype](#)*) – Selects a solution. (input)
- `y` (`double[]`) – Vector of dual variables corresponding to the constraints. (input)

**Groups** *[Solution](#) - [primal](#)*

`Task.putyslice`

```
void putyslice
(soltype whichsol,
 int first,
 int last,
 double[] y)
```

Sets a slice of the  $y$  vector for a solution.

**Parameters**

- `whichsol` (*[soltype](#)*) – Selects a solution. (input)
- `first` (`int`) – First index in the sequence. (input)
- `last` (`int`) – Last index plus 1 in the sequence. (input)
- `y` (`double[]`) – Vector of dual variables corresponding to the constraints. (input)

**Groups** *[Solution](#) - [dual](#)*

`Task.readdata`

```
void readdata (string filename)
```

Reads an optimization problem and associated data from a file.

**Parameters** filename (**string**) – A valid file name. (input)

**Groups** *Input/Output*

Task.readdataformat

```
void readdataformat  
(string filename,  
 int format,  
 int compress)
```

Reads an optimization problem and associated data from a file.

**Parameters**

- filename (**string**) – A valid file name. (input)
- format (*int*) – File data format. (input)
- compress (*int*) – File compression type. (input)

**Groups** *Input/Output*

Task.readjsonstring

```
void readjsonstring (string 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 (**string**) – Problem data in text format. (input)

**Groups** *Input/Output*

Task.readlpstring

```
void readlpstring (string data)
```

Load task data from a string in LP format, replacing any data that already exists in the task object.

**Parameters** data (**string**) – Problem data in text format. (input)

**Groups** *Input/Output*

Task.readopfstring

```
void readopfstring (string data)
```

Load task data from a string in OPF format, replacing any data that already exists in the task object.

**Parameters** data (**string**) – Problem data in text format. (input)

**Groups** *Input/Output*

Task.readparamfile

```
void readparamfile (string 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`.

**Parameters** `filename (string)` – A valid file name. (input)

**Groups** *Input/Output, Parameters*

`Task.readptfstring`

```
void readptfstring (string 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 (string)` – Problem data in text format. (input)

**Groups** *Input/Output*

`Task.readsolution`

```
void readsolution  
  (soltype whichsol,  
   string 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 (soltype)` – Selects a solution. (input)
- `filename (string)` – A valid file name. (input)

**Groups** *Input/Output*

`Task.readsummary`

```
void readsummary (streamtype whichstream)
```

Prints a short summary of last file that was read.

**Parameters** `whichstream (streamtype)` – Index of the stream. (input)

**Groups** *Input/Output, Inspecting the task*

`Task.readtask`

```
void readtask (string 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 (string)` – A valid file name. (input)

**Groups** *Input/Output*

#### Task.removebarvars

```
void removebarvars (int[] subset)
```

```
void removebarvars  
(int num,  
 int[] 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** (int[]) – Indexes of symmetric matrices which should be removed. (input)
- **num** (int) – Number of symmetric matrices which should be removed. (input)

**Groups** *Problem data - semidefinite*

#### Task.removecones

```
void removecones (int[] subset)
```

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** (int[]) – Indexes of cones which should be removed. (input)

**Groups** *Problem data - cones*

#### Task.removecons

```
void removecons (int[] subset)
```

```
void removecons  
(int num,  
 int[] subset)
```

The function removes a subset of the constraints from the optimization task. This implies that the remaining constraints are renumbered.

##### Parameters

- **subset** (int[]) – Indexes of constraints which should be removed. (input)
- **num** (int) – Number of constraints which should be removed. (input)

**Groups** *Problem data - constraints, Problem data - linear part*

#### Task.removevars

```
void removevars (int[] subset)
```

```
void removevars  
(int num,  
 int[] subset)
```

The function removes a subset of the variables from the optimization task. This implies that the remaining variables are renumbered.

##### Parameters

- `subset (int[])` – Indexes of variables which should be removed. (input)
- `num (int)` – Number of variables which should be removed. (input)

**Groups** *Problem data - variables, Problem data - linear part*

`Task.resizetask`

```
void resizetask
(int maxnumcon,
 int maxnumvar,
 int maxnumcone,
 long maxnumanz,
 long 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 (int)` – New maximum number of constraints. (input)
- `maxnumvar (int)` – New maximum number of variables. (input)
- `maxnumcone (int)` – New maximum number of cones. (input)
- `maxnumanz (long)` – New maximum number of non-zeros in  $A$ . (input)
- `maxnumqnz (long)` – New maximum number of non-zeros in all  $Q$  matrices. (input)

**Groups** *Environment and task management*

`Task.sensitivityreport`

```
void sensitivityreport (streamtype 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 (streamtype)` – Index of the stream. (input)

**Groups** *Sensitivity analysis*

`Task.set_InfoCallback`

```
void set_InfoCallback (DataCallback callback)
```

Receive callbacks with solver status and information during optimization.

**Parameters** `callback (DataCallback)` – The callback object. (input)

`Task.set_ItgSolutionCallback`

```
void set_ItgSolutionCallback (ItgSolutionCallback callback)
```

Receive callbacks with solution updates from the mixed-integer optimizer.

**Parameters** `callback (ItgSolutionCallback)` – The callback object. (input)

`Task.set_Progress`

```
void set_Progress (Progress callback)
```

Receive callbacks about current status of the solver during optimization.

**Parameters** callback (*Progress*) – The callback object. (input)

Task.set\_Stream

```
void set_Stream  
(streamtype whichstream,  
 Stream callback)
```

Directs all output from a task stream to a callback object.

**Parameters**

- whichstream (*streamtype*) – Index of the stream. (input)
- callback (*Stream*) – The callback object. (input)

Task.setdefaults

```
void setdefaults ()
```

Resets all the parameters to their default values.

**Groups** *Parameters*

Task.solutiondef

```
int solutiondef (soltype whichsol)
```

```
void solutiondef  
(soltype whichsol,  
 out int isdef)
```

Checks whether a solution is defined.

**Parameters**

- whichsol (*soltype*) – Selects a solution. (input)
- isdef (int) – Is non-zero if the requested solution is defined. (output)

**Return** (int) – Is non-zero if the requested solution is defined.

**Groups** *Solution information*

Task.solutionsummary

```
void solutionsummary (streamtype whichstream)
```

Prints a short summary of the current solutions.

**Parameters** whichstream (*streamtype*) – Index of the stream. (input)

**Groups** *Logging, Solution information*

Task.solvewithbasis

```
void solvewithbasis  
(int transp,  
 ref int numnz,  
 int[] sub,  
 double[] val)
```

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

$$B\bar{X} = b \quad (15.3)$$

or the system

$$B^T\bar{X} = b \quad (15.4)$$

for the unknowns  $\bar{X}$ , with  $b$  being a user-defined vector. In order to make sense of the solution  $\bar{X}$  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

$$B_{i,k} = A_{i,j}, \quad i = 0, \dots, \text{numcon} - 1.$$

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** (int) – If this argument is zero, then (15.3) is solved, if non-zero then (15.4) is solved. (input)
- **numnz** (int) – As input it is the number of non-zeros in  $b$ . As output it is the number of non-zeros in  $\bar{X}$ . (input/output)
- **sub** (int[]) – As input it contains the positions of non-zeros in  $b$ . As output it contains the positions of the non-zeros in  $\bar{X}$ . It must have room for *numcon* elements. (input/output)
- **val** (double[]) – 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  $\text{val}[i] = 0$  when  $b[i] = 0$ ). As output **val** is the vector  $\bar{X}$  as a dense vector. It must have length *numcon*. (input/output)

**Groups** *Solving systems with basis matrix*

**Task.strtoconetype**

```
void strtoconetype
(string str,
 out conetype conetype)
```

Obtains cone type code corresponding to a cone type string.

#### Parameters

- **str** (string) – String corresponding to the cone type code **conetype**. (input)
- **conetype** (*conetype*) – The cone type corresponding to the string **str**. (output)

**Groups** *Names*

**Task.strtosk**

```
void strtosk
(string str,
 out stakey sk)
```

Obtains the status key corresponding to an abbreviation string.

**Parameters**

- **str** (*string*) – A status key abbreviation string. (input)
- **sk** (*stakey*) – Status key corresponding to the string. (output)

**Groups** *Names*

**Task.toconic**

```
void toconic ()
```

This function tries to reformulate a given Quadratically Constrained Quadratic Optimization problem (QCQP) 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**

```
void updatesolutioninfo (soltype whichsol)
```

Update the information items related to the solution.

**Parameters** *whichsol* (*soltype*) – Selects a solution. (input)

**Groups** *Information items and statistics*

**Task.writedata**

```
void writedata (string 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* (*string*) – A valid file name. (input)

**Groups** *Input/Output*

**Task.writejsonsol**

```
void writejsonsol (string filename)
```

Saves the current solutions and solver information items in a JSON file.

**Parameters** filename (string) – A valid file name. (input)

**Groups** *Input/Output*

Task.writeparamfile

```
void writeparamfile (string filename)
```

Writes all the parameters to a parameter file.

**Parameters** filename (string) – A valid file name. (input)

**Groups** *Input/Output*, *Parameters*

Task.writesolution

```
void writesolution  
(soltype whichsol,  
 string filename)
```

Saves the current basic, interior-point, or integer solution to a file.

**Parameters**

- whichsol (*soltype*) – Selects a solution. (input)
- filename (string) – A valid file name. (input)

**Groups** *Input/Output*

Task.writetask

```
void writetask (string 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 (string) – A valid file name. (input)

**Groups** *Input/Output*

## 15.5 Exceptions

MosekException

The base class for all exceptions in **MOSEK**.

**Implements** Exception

Exception

Base class for exceptions that correspond to **MOSEK** response codes.

**Implements** *MosekException*

Error

Exception class used for all error response codes from **MOSEK**.

**Implements** *Exception*

Warning

Exception class used for all warning response codes from **MOSEK**.

**Implements** *MosekException*

**NullPointerException**

Exception thrown when null was passed to a method that expected non-null array argument.

**Implements** *MosekException*

**ArrayLengthException**

Exception thrown the length of an array was smaller than required. This will happen, for example, if requesting a list of  $N$  values, but the array passed to the method is less than  $N$  elements long.

**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*
- *iparam.check\_convexity*
- *iparam.log\_check\_convexity*

## 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\_transform*
- *iparam.read\_debug*
- *iparam.read\_keep\_free\_con*
- *iparam.read\_lp\_drop\_new\_vars\_in\_bou*
- *iparam.read\_lp\_quoted\_names*
- *iparam.read\_mps\_format*

- *iparam.read\_mps\_width*
- *iparam.read\_task\_ignore\_param*
- *iparam.sol\_read\_name\_width*
- *iparam.sol\_read\_width*
- *iparam.write\_bas\_constraints*
- *iparam.write\_bas\_head*
- *iparam.write\_bas\_variables*
- *iparam.write\_compression*
- *iparam.write\_data\_param*
- *iparam.write\_free\_con*
- *iparam.write\_generic\_names*
- *iparam.write\_generic\_names\_io*
- *iparam.write\_ignore\_incompatible\_items*
- *iparam.write\_int\_constraints*
- *iparam.write\_int\_head*
- *iparam.write\_int\_variables*
- *iparam.write\_lp\_full\_obj*
- *iparam.write\_lp\_line\_width*
- *iparam.write\_lp\_quoted\_names*
- *iparam.write\_lp\_strict\_format*
- *iparam.write\_lp\_terms\_per\_line*
- *iparam.write\_mps\_format*
- *iparam.write\_mps\_int*
- *iparam.write\_precision*
- *iparam.write\_sol\_barvariables*
- *iparam.write\_sol\_constraints*
- *iparam.write\_sol\_head*
- *iparam.write\_sol\_ignore\_invalid\_names*
- *iparam.write\_sol\_variables*
- *iparam.write\_task\_inc\_sol*
- *iparam.write\_xml\_mode*
- *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\_file\_name*
- *sparam.stat\_key*
- *sparam.stat\_name*
- *sparam.write\_lp\_gen\_var\_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.check\_convexity\_rel\_tol*
- *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.qcgo\_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\_max\_num\_refinement\_steps*
- *iparam.intpnt\_off\_col\_trh*
- *iparam.intpnt\_order\_gp\_num\_seeds*
- *iparam.intpnt\_order\_method*

- *iparam.intpnt\_purify*
- *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.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\_response*
- *iparam.log\_sensitivity*
- *iparam.log\_sensitivity\_opt*
- *iparam.log\_sim*
- *iparam.log\_sim\_freq*
- *iparam.log\_storage*

## Mixed-integer optimization

- *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\_conic\_outer\_approximation*
- *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\_selection\_level*
- *iparam.mio\_feaspump\_level*
- *iparam.mio\_heuristic\_level*
- *iparam.mio\_max\_num\_branches*
- *iparam.mio\_max\_num\_relaxs*
- *iparam.mio\_max\_num\_root\_cut\_rounds*
- *iparam.mio\_max\_num\_solutions*
- *iparam.mio\_node\_optimizer*
- *iparam.mio\_node\_selection*
- *iparam.mio\_perspective\_reformulate*
- *iparam.mio\_probing\_level*
- *iparam.mio\_propagate\_objective\_constraint*
- *iparam.mio\_rins\_max\_nodes*
- *iparam.mio\_root\_optimizer*
- *iparam.mio\_root\_repeat\_presolve\_level*
- *iparam.mio\_seed*
- *iparam.mio\_ub\_detection\_level*

## Output information

- *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\_response*
- *iparam.log\_sensitivity*
- *iparam.log\_sensitivity\_opt*
- *iparam.log\_sim*
- *iparam.log\_sim\_freq*
- *iparam.log\_sim\_minor*
- *iparam.log\_storage*
- *iparam.max\_num\_warnings*

## Overall solver

- *iparam.bi\_clean\_optimizer*
- *iparam.infeas\_prefer\_primal*
- *iparam.license\_wait*
- *iparam.mio\_mode*
- *iparam.optimizer*
- *iparam.presolve\_level*
- *iparam.presolve\_max\_num\_reductions*
- *iparam.presolve\_use*

- *iparam.primal\_repair\_optimizer*
- *iparam.sensitivity\_all*
- *iparam.sensitivity\_optimizer*
- *iparam.sensitivity\_type*
- *iparam.solution\_callback*

## Overall system

- *iparam.auto\_update\_sol\_info*
- *iparam.intpnt\_multi\_thread*
- *iparam.license\_wait*
- *iparam.log\_storage*
- *iparam.mt\_spincount*
- *iparam.num\_threads*
- *iparam.remove\_unused\_solutions*
- *iparam.timing\_level*
- *sparam.remote\_access\_token*

## Presolve

- *dparam.presolve\_tol\_abs\_lindep*
- *dparam.presolve\_tol\_aij*
- *dparam.presolve\_tol\_rel\_lindep*
- *dparam.presolve\_tol\_s*
- *dparam.presolve\_tol\_x*
- *iparam.presolve\_eliminator\_max\_fill*
- *iparam.presolve\_eliminator\_max\_num\_tries*
- *iparam.presolve\_level*
- *iparam.presolve\_lindep\_abs\_work\_trh*
- *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*

## Progress callback

- *iparam.solution\_callback*

## Simplex optimizer

- *dparam.basis\_rel\_tol\_s*
- *dparam.basis\_tol\_s*
- *dparam.basis\_tol\_x*
- *dparam.sim\_lu\_tol\_rel\_piv*
- *dparam.simplex\_abs\_tol\_piv*
- *iparam.basis\_solve\_use\_plus\_one*
- *iparam.log\_sim*
- *iparam.log\_sim\_freq*
- *iparam.log\_sim\_minor*
- *iparam.sensitivity\_optimizer*
- *iparam.sim\_basis\_factor\_use*
- *iparam.sim\_degen*
- *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\_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\_stability\_priority*
- *iparam.sim\_switch\_optimizer*

## Solution input/output

- *iparam.infeas\_report\_auto*
- *iparam.sol\_filter\_keep\_basic*
- *iparam.sol\_filter\_keep\_ranged*
- *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\_time*
- *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*

## 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.check_convexity_rel_tol`

This parameter controls when the full convexity check declares a problem to be non-convex. Increasing this tolerance relaxes the criteria for declaring the problem non-convex.

A problem is declared non-convex if negative (positive) pivot elements are detected in the Cholesky factor of a matrix which is required to be PSD (NSD). This parameter controls how much this non-negativity requirement may be violated.

If  $d_i$  is the pivot element for column  $i$ , then the matrix  $Q$  is considered to not be PSD if:

$$d_i \leq -|Q_{ii}| \text{check\_convexity\_rel\_tol}$$

**Default** 1e-10

**Accepted** [0; +inf]

**Example** `task.putdouparam(dparam.check_convexity_rel_tol, 1e-10)`

**Generic name** MSK\_DPAR\_CHECK\_CONVEXITY\_REL\_TOL

**Groups** *Interior-point method*









**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*







**Default** 1.0e-8  
**Accepted** [0.0; +inf]  
**Example** `task.putdoupparam(dparam.presolve_tol_x, 1.0e-8)`  
**Generic name** MSK\_DPAR\_PREOLVE\_TOL\_X  
**Groups** *Presolve*

`dparam.qcqp_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.putdoupparam(dparam.qcqp_reformulate_rel_drop_tol, 1e-15)`  
**Generic name** MSK\_DPAR\_QCQP\_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.putdoupparam(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.putdoupparam(dparam.sim_lu_tol_rel_piv, 0.01)`  
**Generic name** MSK\_DPAR\_SIM\_LU\_TOL\_REL\_PIV  
**Groups** *Basis identification, Simplex optimizer*

`dparam.simplex_abs_tol_piv`

Absolute pivot tolerance employed by the simplex optimizers.

**Default** 1.0e-7  
**Accepted** [1.0e-12; +inf]  
**Example** `task.putdoupparam(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** 1.0e30  
**Accepted** [-inf; +inf]  
**Example** `task.putdoupparam(dparam.upper_obj_cut, 1.0e30)`  
**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  $\infty$ .

**Default** 0.5e30  
**Accepted** [-inf; +inf]  
**Example** task.putdoupparam(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, 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.check_convexity`  
Specify the level of convexity check on quadratic problems.

**Default** *full*  
**Accepted** *none, simple, full* (see *checkconvexitytype*)  
**Example** `task.putintparam(iparam.check_convexity, checkconvexitytype.full)`  
**Generic name** `MSK_IPAR_CHECK_CONVEXITY`  
**Groups** *Data check*

`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.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_prefer_primal`  
If both certificates of primal and dual infeasibility are supplied then only the primal is used when this option is turned on.

**Default** *on*  
**Accepted** *on, off* (see *onoffkey*)  
**Example** `task.putintparam(iparam.infeas_prefer_primal, onoffkey.on)`  
**Generic name** `MSK_IPAR_INFEAS_PREFER_PRIMAL`  
**Groups** *Overall solver*

`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`







**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



**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*































**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, moderate, aggressive* (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*)























**rescode.wrn\_lp\_old\_quad\_format (80)**  
 Missing  $\sqrt{2}$  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 is not valid. For instance when writing an LP file the names must not contain blanks or start with a digit.

**rescode.wrn\_license\_feature\_expire (505)**  
 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 an 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\_write\_changed\_names (803)**  
Some names were changed because they were invalid for the output file format.

**rescode.wrn\_write\_discarded\_cfix (804)**  
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  $+\text{inf}$  or  $-\text{inf}$ .

**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\_quad\_cones\_with\_root\_fixed\_at\_zero (930)**  
For at least one quadratic cone the root is fixed at (nearly) zero. This may cause problems such as a very large dual solution. Therefore, it is recommended to remove such cones before optimizing the problem, or to fix all the variables in the cone to 0.

**rescode.wrn\_rquad\_cones\_with\_root\_fixed\_at\_zero (931)**  
For at least one rotated quadratic cone at least one of the root variables are fixed at (nearly) zero. This may cause problems such as a very large dual solution. Therefore, it is recommended to remove such cones before optimizing the problem, or to fix all the variables in the cone to 0.

**rescode.wrn\_exp\_cones\_with\_variables\_fixed\_at\_zero (932)**  
For at least one exponential cone  $x \geq y \exp(z/y)$  either the variable  $x$  or  $y$  is fixed at (nearly) zero. This may cause problems such as a very large dual solution. Therefore, it is recommended to remove such cones before optimizing the problem, or to fix all the variables in the cone to 0.

**rescode.wrn\_pow\_cones\_with\_root\_fixed\_at\_zero (933)**  
For at least one power cone at least one of the root variables are fixed at (nearly) zero. This may cause problems such as a very large dual solution. Therefore, it is recommended to remove such cones before optimizing the problem, or to fix all the variables in the cone to 0.

**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.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 reached.

rescode.err\_null\_env (1060)  
env is a NULL pointer.

rescode.err\_null\_task (1061)  
task is a NULL 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 NULL pointer.

rescode.err\_living\_tasks (1066)  
All tasks associated with an environment must be deleted before the environment is deleted. There are still some undeleted tasks.

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_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_read_format (1090)`  
 The specified format cannot be read.

`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_inv_con_key (1107)`  
 An invalid constraint key occurred in an MPS file.

`rescode.err_mps_inv_bound_key (1108)`  
 An invalid bound key occurred in an MPS file.

`rescode.err_mps_inv_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_splitting_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 QSECTIONs 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 CSECTIONs 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 specified 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_lp_incompatible (1150)`  
 The problem cannot be written to an LP formatted file.

`rescode.err_lp_empty (1151)`  
 The problem cannot be written to an LP formatted file.

`rescode.err_lp_dup_slack_name (1152)`  
 The name of the slack variable added to a ranged constraint already exists.

`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_lp_free_constraint (1155)`  
 Free constraints cannot be written in LP file format.

`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_write_lp_format (1158)`  
 Problem cannot be written as an LP file.

`rescode.err_read_lp_missing_end_tag (1159)`  
 Syntax error in LP file. Possibly missing End tag.

`rescode.err_lp_format (1160)`  
 Syntax error in an LP file.

`rescode.err_write_lp_non_unique_name (1161)`  
 An auto-generated name is not unique.

`rescode.err_read_lp_nonexisting_name (1162)`  
 A variable never occurred in objective or constraints.

`rescode.err_lp_write_conic_problem (1163)`  
 The problem contains cones that cannot be written to an LP formatted file.

`rescode.err_lp_write_geco_problem (1164)`  
 The problem contains general convex terms that cannot be written to an LP formatted file.

`rescode.err_writing_file (1166)`  
 An error occurred while writing file

`rescode.err_ptf_format (1167)`  
 Syntax error in an PTF file

`rescode.err_opf_format (1168)`  
 Syntax error in an OPF file

`rescode.err_opf_new_variable (1169)`  
 Introducing new variables is now allowed. When a `[variables]` section is present, it is not allowed to introduce new variables later in the problem.

`rescode.err_invalid_name_in_sol_file (1170)`  
 An invalid name occurred in a solution file.

`rescode.err_lp_invalid_con_name (1171)`  
 A constraint name is invalid when used in an LP formatted file.

**rescode.err\_opf\_premature\_eof (1172)**  
 Premature end of file in an OPF 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\_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\_param\_name (1205)**  
 The parameter name is not correct.

**rescode.err\_param\_name\_dou (1206)**  
 The parameter name is not correct for a double parameter.

**rescode.err\_param\_name\_int (1207)**  
 The parameter name is not correct for an integer parameter.

**rescode.err\_param\_name\_str (1208)**  
 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.snz` 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_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_proptype` (1259)  
 Problem type does not match the chosen optimizer.

`rescode.err_objective_range` (1260)  
 Empty objective range.

`rescode.err_undef_solution` (1265)  
**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_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_skc` (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. The parameter `dparam.check_convexity_rel_tol` can be used to relax the convexity check.

`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. The parameter `dparam.check_convexity_rel_tol` can be used to relax the convexity check.

`rescode.err_obj_q_not_psd (1295)`  
The quadratic coefficient matrix in the objective is not positive semidefinite as expected for a minimization problem. The parameter `dparam.check_convexity_rel_tol` can be used to relax the convexity check.

`rescode.err_obj_q_not_nsd (1296)`  
The quadratic coefficient matrix in the objective is not negative semidefinite as expected for a maximization problem. The parameter `dparam.check_convexity_rel_tol` can be used to relax the convexity check.

`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).

`rescode.err_upper_bound_is_a_nan (1391)`  
The upper bound specified is not a number (nan).

`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 truncation 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_nan_in_blc` (1461)  
 $l^c$  contains an invalid floating point value, i.e. a NaN.

`rescode.err_nan_in_buc` (1462)  
 $u^c$  contains an invalid floating point value, i.e. a NaN.

`rescode.err_nan_in_c` (1470)  
 $c$  contains an invalid floating point value, i.e. a NaN.

`rescode.err_nan_in_blx` (1471)  
 $l^x$  contains an invalid floating point value, i.e. a NaN.

`rescode.err_nan_in_bux` (1472)  
 $u^x$  contains an invalid floating point value, i.e. a NaN.

`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 NULL 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_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_xml_invalid_problem_type (3600)`  
The problem type is not supported by the XML format.

`rescode.err_invalid_ampl_stub (3700)`  
Invalid AMPL stub.

`rescode.err_int64_to_int32_cast (3800)`  
A 64 bit integer could not be cast to a 32 bit integer.

`rescode.err_size_license_numcores (3900)`  
The computer contains more cpu cores than the license allows for.

`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  $\bar{X}$  available for the solution specified. In particular note there are no  $\bar{X}$  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_triangular (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 matrix has 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 conic constraints.

`rescode.err_invalid_file_format_for_nonlinear (4010)`  
The file format does not support a problem with nonlinear terms.

`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_non_unique_array (5000)`  
 An array does not contain unique elements.

`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 specified. Note only the lower triangular part with no duplicates is specified.

`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 (7109)
    Duplicate VAR keyword.
rescode.err_cbf_duplicate_int (7110)
    Duplicate INT keyword.
rescode.err_cbf_invalid_var_type (7111)
    Invalid variable type.
rescode.err_cbf_invalid_con_type (7112)
    Invalid constraint type.
rescode.err_cbf_invalid_domain_dimension (7113)
    Invalid domain dimension.
rescode.err_cbf_duplicate_objacoord (7114)
    Duplicate index in OBJCOORD.
rescode.err_cbf_duplicate_bcoord (7115)
    Duplicate index in BCOORD.
rescode.err_cbf_duplicate_acoord (7116)
    Duplicate index in ACOORD.
rescode.err_cbf_too_few_variables (7117)
    Too few variables defined.
rescode.err_cbf_too_few_constraints (7118)
    Too few constraints defined.
rescode.err_cbf_too_few_ints (7119)
    Too few ints are specified.
rescode.err_cbf_too_many_ints (7120)
    Too many ints are specified.
rescode.err_cbf_invalid_int_index (7121)
    Invalid INT index.
rescode.err_cbf_unsupported (7122)
    Unsupported feature is present.
rescode.err_cbf_duplicate_psdvar (7123)
    Duplicate PSDVAR keyword.
rescode.err_cbf_invalid_psdvar_dimension (7124)
    Invalid PSDVAR dimension.
rescode.err_cbf_too_few_psdvar (7125)
    Too few variables defined.
rescode.err_cbf_invalid_exp_dimension (7126)
    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 its definition.

`rescode.err_cbf_invalid_number_of_cones` (7740)  
Invalid number of cones.

`rescode.err_cbf_invalid_dimension_of_cones` (7741)  
Invalid dimension of cones.

`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_toconic_constr_q_not_psd` (7800)  
The matrix defining the quadratic 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 quadratic part of the objective function is not positive semidefinite.

`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_problem_size` (8008)  
The size of the problem exceeds the dimensions permitted by the instance of the OptServer where it was run.

## 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 a 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.

`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 optimizer 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.

`purify`  
Solution purification employed optimizer.

`purify.none`  
The optimizer performs no solution purification.

`purify.primal`  
The optimizer purifies the primal solution.

`purify.dual`  
The optimizer purifies the dual solution.

`purify.primal_dual`  
The optimizer purifies both the primal and dual solution.

`purify.auto`  
TBD

`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_full_convexity_check`  
Begin full convexity check.

`callbackcode.begin_infeas_ana`  
The callback function is called when the infeasibility analyzer 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_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_qcqp_reformulate`  
Begin QCQP 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_simplex_bi`  
The callback function is called from within the basis identification procedure when the simplex clean-up phase 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.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_full_convexity_check`  
End full convexity check.

`callbackcode.end_infeas_ana`  
The callback function is called when the infeasibility analyzer 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_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_qcqp_reformulate`  
End QCQP 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_to_conic`  
End conic reformulation.

`callbackcode.end_write`  
**MOSEK** has finished writing a problem file.

`callbackcode.im_bi`  
The callback function is called from within the basis identification procedure at an intermediate point.

`callbackcode.im_conic`  
The callback function is called at an intermediate stage within the conic optimizer where the information database has not been updated.

`callbackcode.im_dual_bi`  
The callback function is called from within the basis identification procedure at an intermediate point in the dual phase.

`callbackcode.im_dual_sensitivity`  
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_full_convexity_check`  
The callback function is called at an intermediate stage of the full convexity check.

`callbackcode.im_intpnt`  
The callback function is called at an intermediate stage within the interior-point optimizer where the information database has not been updated.

`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_presolve`  
The callback function is called from within the presolve procedure at an intermediate stage.

`callbackcode.im_primal_bi`  
The callback function is called from within the basis identification procedure at an intermediate point in the primal phase.

`callbackcode.im_primal_sensitivity`  
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_qo_reformulate`  
The callback function is called at an intermediate stage of the conic quadratic reformulation.

`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.im_simplex_bi`  
The callback function is called from within the basis identification procedure at an intermediate point in the simplex clean-up phase. The frequency of the callbacks is controlled by the `iparam.log_sim_freq` parameter.

`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.primal_simplex`  
The callback function is called from within the primal simplex optimizer.

`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.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.write_opf`  
The callback function is called from the OPF writer.

`checkconvexitytype`  
Types of convexity checks.

`checkconvexitytype.none`  
No convexity check.

`checkconvexitytype.simple`  
Perform simple and fast convexity check.

`checkconvexitytype.full`  
 Perform a full convexity check.

`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.

`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.

`scopr`  
 SCopt operator types

`scopr.ent`  
 Entropy

`scopr.exp`  
 Exponential

`scopr.log`  
 Logarithm

`scopr.pow`  
 Power

`scopr.sqrt`  
 Square root

`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.

**dinfitem**  
 Double information items

**dinfitem.bi\_clean\_dual\_time**  
 Time spent within the dual clean-up optimizer of the basis identification procedure since its invocation.

**dinfitem.bi\_clean\_primal\_time**  
 Time spent within the primal clean-up optimizer of the basis identification procedure since its invocation.

**dinfitem.bi\_clean\_time**  
 Time spent within the clean-up phase of the basis identification procedure since its invocation.

**dinfitem.bi\_dual\_time**  
 Time spent within the dual phase basis identification procedure since its invocation.

**dinfitem.bi\_primal\_time**  
 Time spent within the primal phase of the basis identification procedure since its invocation.

**dinfitem.bi\_time**  
 Time spent within the basis identification procedure since its invocation.

**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.

`dinfitem.mio_clique_separation_time`  
 Separation time for clique cuts.

`dinfitem.mio_cmir_separation_time`  
 Separation time for CMIR cuts.

`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_separation_time`  
 Separation time for GMI cuts.

`dinfitem.mio_implied_bound_time`  
 Separation time for implied bound cuts.

`dinfitem.mio_knapsack_cover_separation_time`  
 Separation time for knapsack cover.

`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 `dinfitem.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 `dinfitem.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  $\delta$  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.

`dinfitem.mio_root_cutgen_time`  
 Total time for cut generation.

`dinfitem.mio_root_optimizer_time`  
 Time spent in the optimizer while solving the root node relaxation

`dinfitem.mio_root_presolve_time`  
Time spent presolving the problem at the root node.

`dinfitem.mio_time`  
Time spent in the mixed-integer optimizer.

`dinfitem.mio_user_obj_cut`  
If the objective cut is used, then this information item has the value of the cut.

`dinfitem.optimizer_time`  
Total time spent in the optimizer since it was invoked.

`dinfitem.presolve_eli_time`  
Total time spent in the eliminator since the presolve was invoked.

`dinfitem.presolve_lindep_time`  
Total time spent in the linear dependency checker since the presolve was invoked.

`dinfitem.presolve_time`  
Total time (in seconds) spent in the presolve since it was invoked.

`dinfitem.primal_repair_penalty_obj`  
The optimal objective value of the penalty function.

`dinfitem.qcqp_reformulate_max_perturbation`  
Maximum absolute diagonal perturbation occurring during the QCQP reformulation.

`dinfitem.qcqp_reformulate_time`  
Time spent with conic quadratic reformulation.

`dinfitem.qcqp_reformulate_worst_cholesky_column_scaling`  
Worst Cholesky column scaling.

`dinfitem.qcqp_reformulate_worst_cholesky_diag_scaling`  
Worst Cholesky diagonal scaling.

`dinfitem.rd_time`  
Time spent reading the data file.

`dinfitem.sim_dual_time`  
Time spent in the dual simplex optimizer since invoking it.

`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.

`dinfitem.sim_time`  
Time spent in the simplex optimizer since invoking it.

`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.update\_solutioninfo*.

`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.update\_solutioninfo*.

`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.update\_solutioninfo*.

`dinfitem.sol_bas_nrm_barx`  
Infinity norm of  $\bar{X}$  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.update_solution_info`.

`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.update_solution_info`.

`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.update_solution_info`.

`dinfitem.sol_itg_nrm_barx`  
 Infinity norm of  $\bar{X}$  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.update_solution_info`.

`dinfitem.sol_itg_pviolbarvar`  
 Maximal primal bound violation for  $\bar{X}$  in the integer solution. Updated if `iparam.auto_update_sol_info` is set or by the method `Task.update_solution_info`.

`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.update_solution_info`.

`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.update_solution_info`.

`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.update_solution_info`.

`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.update_solution_info`.

`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.update_solution_info`.

`dinfitem.sol_itr_dviolbarvar`  
 Maximal dual bound violation for  $\bar{X}$  in the interior-point solution. Updated if `iparam.auto_update_sol_info` is set or by the method `Task.update_solution_info`.

`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.update_solution_info`.

`dinfitem.sol_itr_dviolcones`  
Maximal dual violation for dual conic constraints in the interior-point solution. Updated if `iparam.auto_update_sol_info` is set or by the method `Task.update_solution_info`.

`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.update_solution_info`.

`dinfitem.sol_itr_nrm_bars`  
Infinity norm of  $\bar{S}$  in the interior-point solution.

`dinfitem.sol_itr_nrm_barx`  
Infinity norm of  $\bar{X}$  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.update_solution_info`.

`dinfitem.sol_itr_pviolbarvar`  
Maximal primal bound violation for  $\bar{X}$  in the interior-point solution. Updated if `iparam.auto_update_sol_info` is set or by the method `Task.update_solution_info`.

`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.update_solution_info`.

`dinfitem.sol_itr_pviolcones`  
Maximal primal violation for primal conic constraints in the interior-point solution. Updated if `iparam.auto_update_sol_info` is set or by the method `Task.update_solution_info`.

`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.update_solution_info`.

`dinfitem.to_conic_time`  
Time spent in the last to conic reformulation.

`feature`  
License feature

`feature.pts`  
Base system.

`feature.pton`  
Conic extension.

`liinfitem`  
Long integer information items.

`liinfitem.bi_clean_dual_deg_iter`  
Number of dual degenerate clean iterations performed in the basis identification.

`liinfitem.bi_clean_dual_iter`  
Number of dual clean iterations performed in the basis identification.

`liinfitem.bi_clean_primal_deg_iter`  
Number of primal degenerate clean iterations performed in the basis identification.

`liinfitem.bi_clean_primal_iter`  
Number of primal 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.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_intpnt_iter`  
Number of interior-point iterations performed 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_numanz`  
Number of non-zeros in A that is read.

`liinfitem.rd_numqnz`  
Number of Q non-zeros.

`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.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_node_depth`  
Depth of the last node solved.

`iinfitem.mio_num_active_nodes`  
Number of active branch and bound nodes.

`iinfitem.mio_num_branch`  
Number of branches performed during the optimization.

`iinfitem.mio_num_clique_cuts`  
Number of clique cuts.

`iinfitem.mio_num_cmir_cuts`  
Number of Complemented Mixed Integer Rounding (CMIR) cuts.

`iinfitem.mio_num_gomory_cuts`  
Number of Gomory cuts.

`iinfitem.mio_num_implied_bound_cuts`  
Number of implied bound cuts.

`iinfitem.mio_num_int_solutions`  
Number of integer feasible solutions that have been found.

`iinfitem.mio_num_knapsack_cover_cuts`  
Number of clique cuts.

`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_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_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_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_cuts`  
Total number of cuts generated 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.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\_prototype**  
 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 indicate that memory fragmentation may occur.

**infotype**  
 Information item types

**infotype.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.

**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.

**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.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.

`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.

`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.

`problemtypes`  
Problem types

`problemtypes.lo`  
The problem is a linear optimization problem.

`problemtypes.qo`  
The problem is a quadratic optimization problem.

`problemtypes.qcqp`  
The problem is a quadratically constrained optimization problem.

`problemtypes.conic`  
A conic optimization.

`problemtypes.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.

**xmlwriteroutputtype**  
 XML writer output mode

**xmlwriteroutputtype.row**  
 Write in row order.

**xmlwriteroutputtype.col**  
 Write in column order.

**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.

**scalingtype.moderate**  
 A conservative scaling is performed.

**scalingtype.aggressive**  
 A very aggressive 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.

`startpointtype.satisfy_bounds`  
The starting point is chosen to satisfy all the simple bounds on nonlinear variables. If this starting point is employed, then more care than usual should be employed when choosing the bounds on the nonlinear variables. In particular very tight bounds should be avoided.

`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 Class types

`mosek.Progress`

A handler class for progress call backs.

`Progress.progressCB`

```
int progressCB (callbackcode code)
```

The progress callback 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.

**Parameters** `code` (*callbackcode*) – Callback code indicating current operation of the solver. (input)

**Return** (`int`) – Non-zero if the optimizer should be stopped; zero otherwise.

`mosek.ItgSolutionCallback`

A handler class for integer solution call backs.

`ItgSolutionCallback.callback`

```
void callback (double [] xx)
```

The integer solution callback is a user-defined function which will be called by **MOSEK** when it improves the best mixed-integer solution.

The user *must not* call any **MOSEK** function directly or indirectly from the callback function.

**Parameters** `xx (double[])` – An array with the values of all variables in the currently best solution. (input)

`mosek.DataCallback`

A handler class for data call backs.

`DataCallback.callback`

```
int callback
(callbackcode code,
 double [] dinf,
 int [] iinf,
 long [] liinf)
```

The data callback 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 (double[])` – Array of double information items. (input)
- `iinf (int[])` – Array of integer information items. (input)
- `liinf (long[])` – Array of long integer information items. (input)

**Return** (`int`) – Non-zero if the optimizer should be stopped; zero otherwise.

`mosek.Stream`

A stream handler class.

`Stream.streamCB`

```
void streamCB (string 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 (string)` – Text string in the stream. (input)

## 15.11 Nonlinear interfaces (obsolete)

---

**Important:** This is a legacy document for users familiar with SCopt, DGopt, EXPopt, mskenopt, mskscpt and mskgpopt from previous versions of **MOSEK**. These interfaces have now been removed. We assume familiarity with documentation included in version 8. All problems expressible with this interface can (and should) be reformulated using the exponential cone and power cones.

New users should formulate problems involving powers, logarithms and exponentials directly in conic form.

---

### Conversion tutorial

We recommend converting all nonlinear problems using SCopt, DGopt, EXPopt, mskenopt, mskscpt and mskgpopt into conic form. Depending on the values of  $f, g, h$  either the epigraph or hypograph of a SCopt function is convex, and a bounding variable can be introduced following the basic rules below. We assume all variables are within safe bounds where the SCopt operators are defined and convex. We also assume  $f > 0$ .

A more comprehensive modeling guide for these types of problems can be found in the **MOSEK Modeling Cookbook**.

### Powers

Consider  $f(x+h)^g$ . This can be reformulated using the power cone.

- If  $g > 1$  then  $t \geq f(x+h)^g$  is equivalent to  $(t/f)^{1/g} \geq |x+h|$ , that is  $(t/f, 1, x+h) \in \mathcal{P}_3^{1/g, 1-1/g}$ .
- If  $0 < g < 1$  then  $|t| \leq f(x+h)^g$  is equivalent to  $(x+h, 1, t/f) \in \mathcal{P}_3^{g, 1-g}$ .
- If  $g < 0$  then  $t \geq f(x+h)^g$  is equivalent to  $(t/f)(x+h)^{-g} \geq 1$ , that is  $(t/f, x+h, 1) \in \mathcal{P}_3^{1/(1-g), -g/(1-g)}$ .

### Logarithm

The bound  $t \leq f \log(gx+h)$  is equivalent to  $(gx+h, 1, t/f) \in K_{\text{exp}}$ .

### Entropy

The bound  $t \geq fx \log x$  is equivalent to  $(1, x, -t/f) \in K_{\text{exp}}$ .

### Exponential

The bound  $t \geq f \exp(gx+h)$  is equivalent to  $(t/f, 1, gx+h) \in K_{\text{exp}}$ .

### Exponential optimization (EXPopt), Geometric programming (mskgpopt)

For a basic tutorial in geometric programming (GP) see [Sec. 6.8](#).

An exponential optimization problem in standard form consists of constraints of the type:

$$t \geq \log \left( \sum_i \exp(a_i^T x + b_i) \right).$$

This log-sum-exp bound is equivalent to

$$\sum_i \exp(a_i^T x + b_i - t) \leq 1$$

and requires bounding each exponential function as explained above.

### Dual geometric optimization (DGopt)

The objective function of a dual geometric problem involves maximizing expressions of the form

$$x \log \frac{c}{x} \quad \text{and} \quad x_i \log \frac{e^T x}{x_i},$$

which can be achieved using bounds  $t \leq x \log \frac{y}{x}$ , that is  $(t, x, y) \in K_{\text{exp}}$ .

## Chapter 16

# Supported File Formats

**MOSEK** supports a range of problem and solution formats listed in [Table 16.1](#) and [Table 16.2](#). The **Task format** is **MOSEK**'s native binary format and it supports all features that **MOSEK** supports. The **OPF format** is **MOSEK**'s human-readable alternative that supports nearly all features (everything except semidefinite problems). In general, text formats are significantly slower to read, but can be examined and edited directly in any text editor.

### Problem formats

Table 16.1: List of supported file formats for optimization problems. The column *Conic* refers to conic problems involving the quadratic, rotated quadratic, power or exponential cone. The last two columns indicate if the format supports solutions and optimizer parameters.

Format Type	Ext.	Binary/Text	LP	QO	Conic	SDP	Sol	Param
<i>LP</i>	lp	plain text	X	X				
<i>MPS</i>	mps	plain text	X	X	X			
<i>OPF</i>	opf	plain text	X	X	X		X	X
<i>PTF</i>	ptf	plain text	X	X	X	X	X	
<i>CBF</i>	cbf	plain text	X		X	X		
<i>Task format</i>	task	binary	X	X	X	X	X	X
<i>Jtask format</i>	jtask	text	X	X	X	X	X	X

### Solution formats

Table 16.2: List of supported solution formats.

Format Type	Ext.	Binary/Text	Description
<i>SOL</i>	sol	plain text	Interior Solution
	bas	plain text	Basic Solution
	int	plain text	Integer
<i>Jsol format</i>	jsol	text	Solution

## 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.gz`

will be considered as a GZIP 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

$$\begin{array}{ll} \text{minimize/maximize} & c^T x + \frac{1}{2} q^o(x) \\ \text{subject to} & \begin{array}{ll} l^c \leq Ax + \frac{1}{2} q(x) \leq u^c, \\ l^x \leq x \leq u^x, \\ x_{\mathcal{J}} \text{ integer,} \end{array} \end{array}$$

where

- $x \in \mathbb{R}^n$  is the vector of decision variables.
- $c \in \mathbb{R}^n$  is the linear term in the objective.
- $q^o : \mathbb{R}^n \rightarrow \mathbb{R}$  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.
- $q : \mathbb{R}^n \rightarrow \mathbb{R}$  is a vector of quadratic functions. Hence,

$$q_i(x) = x^T Q^i x$$

where it is assumed that

$$Q^i = (Q^i)^T.$$

- $\mathcal{J} \subseteq \{1, 2, \dots, n\}$  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.

$$c^T x + \frac{1}{2} x^T Q x.$$

The objective may be given a name by writing

```
myname:
```

before the expressions. If no name is given, then the objective is named **obj**.

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

$$\text{minimize } 4x_1 + x_2 - 0.1 \cdot x_3 + \frac{1}{2}(x_1^2 + 2.1 \cdot x_1 \cdot x_2)$$

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  $\leq$ ) may be any of  $<$ ,  $\leq$ ,  $=$ ,  $>$ ,  $\geq$  ( $<$  and  $\leq$  mean the same), and the bound may be any number.

In the standard LP format it is not possible to define more than one bound per line, but **MOSEK** supports defining ranged constraints by using double-colon ( $::$ ) instead of a single-colon ( $:$ ) after the constraint name, i.e.

$$-5 \leq x_1 + x_2 \leq 5 \quad (16.1)$$

may be written as

```
con:: -5 < x_1 + x_2 < 5
```

By default **MOSEK** writes ranged constraints this way.

If the files must adhere to the LP standard, ranged constraints must either be split into upper bounded and lower bounded constraints or be written as an equality with a slack variable. For example the expression (16.1) may be written as

$$x_1 + x_2 - sl_1 = 0, \quad -5 \leq sl_1 \leq 5.$$

## 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
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.

#### MOSEK Extensions to the LP Format

Some optimization software packages employ a more strict definition of the LP format than the one used by **MOSEK**. The limitations imposed by the strict LP format are the following:

- Quadratic terms in the constraints are not allowed.
- Names can be only 16 characters long.
- Lines must not exceed 255 characters in length.

To get around some of the inconveniences converting from other problem formats, **MOSEK** allows lines to contain 1024 characters and names may have any length (shorter than the 1024 characters).

If an LP formatted file created by **MOSEK** should satisfy the strict definition, then the parameter *iparam.write\_lp\_strict\_format* should be set; note, however, that some problems cannot be written correctly as a strict LP formatted file. For instance, all names are truncated to 16 characters and hence they may lose their uniqueness and change the problem.

Internally in **MOSEK** names may contain any (printable) character, many of which cannot be used in LP names. Setting the parameters *iparam.read\_lp\_quoted\_names* and *iparam.write\_lp\_quoted\_names* allows **MOSEK** to use quoted names. The first parameter tells **MOSEK** to remove quotes from quoted names e.g. "x1", when reading LP formatted files. The second parameter tells **MOSEK** to put quotes around any semi-illegal name (names beginning with a number or a period) and fully illegal name (containing illegal characters). As double quote is a legal character in the LP format, quoting semi-illegal names makes them legal in the pure LP format as long as they are still shorter than 16 characters. Fully illegal names are still illegal in a pure LP file.

### The strict LP format

The LP format is not a formal standard and different vendors have slightly different interpretations of the LP format. To make **MOSEK**'s definition of the LP format more compatible with the definitions of other vendors set the parameter `iparam.write_lp_strict_format` to `onoffkey.on`.

This setting may lead to truncation of some names and hence to an invalid LP file. The simple solution to this problem is to set the parameter `iparam.write_generic_names` to `onoffkey.on` which will cause all names to be renamed systematically in the output file.

### Formatting of an LP File

A few parameters control the visual formatting of LP files written by **MOSEK** in order to make it easier to read the files. These parameters are

- `iparam.write_lp_line_width` sets the maximum number of characters on a single line. The default value is 80 corresponding roughly to the width of a standard text document.
- `iparam.write_lp_terms_per_line` sets the maximum number of terms per line; a term means a sign, a coefficient, and a name (for example `+ 42 elephants`). The default value is 0, meaning that there is no maximum.

### Unnamed Constraints

Reading and writing an LP file with **MOSEK** may change it superficially. If an LP file contains unnamed constraints or objective these are given their generic names when the file is read (however unnamed constraints in **MOSEK** are written without names).

## 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

$$\begin{aligned} \text{maximize/minimize} \quad & c^T x + q_0(x) \\ l^c \leq \quad & Ax + q(x) \leq u^c, \\ l^x \leq \quad & x \leq u^x, \\ & x \in \mathcal{K}, \\ & x_{\mathcal{J}} \text{ integer}, \end{aligned} \tag{16.2}$$

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.
- $q : \mathbb{R}^n \rightarrow \mathbb{R}$  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.
- $\mathcal{J} \subseteq \{1, 2, \dots, n\}$  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

$[+|-]XXXXXXXX.XXXXXX[e|E][+|-]XXX$

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	$\infty$
L (lower)	$-\infty$	finite
N (none)	$-\infty$	$\infty$

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  $i$ -th 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  $Q_{kj}^i$  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

$$\begin{array}{ll} \text{minimize} & -x_2 + \frac{1}{2}(2x_1^2 - 2x_1x_3 + 0.2x_2^2 + 2x_3^2) \\ \text{subject to} & x_1 + x_2 + x_3 \geq 1, \\ & x \geq 0 \end{array}$$

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  $1/2x^T Qx$ .

The example

$$\begin{array}{ll} \text{minimize} & -x_2 + \frac{1}{2}(2x_1^2 - 2x_1x_3 + 0.2x_2^2 + 2x_3^2) \\ \text{subject to} & x_1 + x_2 + x_3 \geq 1, \\ & x \geq 0 \end{array}$$

has the following MPS file representation 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
```

(continues on next page)

(continued from previous page)

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.

### QMATRIX (optional)

A QMATRIX section allows to specify the quadratic part of a given constraint. Within the QMATRIX 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  $Q_{kj}^i$  is assigned the value given by [value1]. Moreover, the quadratic term is represented as  $1/2x^T Qx$ .

The example

$$\begin{aligned}
 &\text{minimize} && x_2 \\
 &\text{subject to} && x_1 + x_2 + x_3 \geq 1, \\
 &&& \frac{1}{2}(-2x_1x_3 + 0.2x_2^2 + 2x_3^2) \leq 10, \\
 &&& x \geq 0
 \end{aligned}$$

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

(continues on next page)

```

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	$-\infty$	$\infty$	No
FX	$v_1$	$v_1$	No
LO	$v_1$	unchanged	No
MI	$-\infty$	unchanged	No
PL	unchanged	$\infty$	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 (16.2). It is assumed that  $\mathcal{K}$  satisfies the following requirements. Let

$$x^t \in \mathbb{R}^{n^t}, \quad t = 1, \dots, k$$

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

$$\mathcal{K} := \{x \in \mathbb{R}^n : x^t \in \mathcal{K}_t, \quad t = 1, \dots, k\}$$

where  $\mathcal{K}_t$  must have one of the following forms:

- $\mathbb{R}$  set:

$$\mathcal{K}_t = \mathbb{R}^{n^t}.$$

- Zero cone:

$$\mathcal{K}_t = \{0\} \subseteq \mathbb{R}^{n^t}. \quad (16.3)$$

- Quadratic cone:

$$\mathcal{K}_t = \left\{ x \in \mathbb{R}^{n^t} : x_1 \geq \sqrt{\sum_{j=2}^{n^t} x_j^2} \right\}. \quad (16.4)$$

- Rotated quadratic cone:

$$\mathcal{K}_t = \left\{ x \in \mathbb{R}^{n^t} : 2x_1x_2 \geq \sum_{j=3}^{n^t} x_j^2, \quad x_1, x_2 \geq 0 \right\}. \quad (16.5)$$

- Primal exponential cone:

$$\mathcal{K}_t = \{x \in \mathbb{R}^3 : x_1 \geq x_2 \exp(x_3/x_2), \quad x_1, x_2 \geq 0\}. \quad (16.6)$$

- Primal power cone (with parameter  $0 < \alpha < 1$ ):

$$\mathcal{K}_t = \left\{ x \in \mathbb{R}^{n^t} : x_1^\alpha x_2^{1-\alpha} \geq \sqrt{\sum_{j=3}^{n^t} x_j^2}, \quad x_1, x_2 \geq 0 \right\}. \quad (16.7)$$

- Dual exponential cone:

$$\mathcal{K}_t = \{x \in \mathbb{R}^3 : x_1 \geq -x_3 e^{-1} \exp(x_2/x_3), \quad x_3 \leq 0, x_1 \geq 0\}. \quad (16.8)$$

- Dual power cone (with parameter  $0 < \alpha < 1$ ):

$$\mathcal{K}_t = \left\{ x \in \mathbb{R}^{n^t} : \left(\frac{x_1}{\alpha}\right)^\alpha \left(\frac{x_2}{1-\alpha}\right)^{1-\alpha} \geq \sqrt{\sum_{j=3}^{n^t} x_j^2}, \quad x_1, x_2 \geq 0 \right\}. \quad (16.9)$$

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

$$2x_3x_7 \geq x_1^2 + x_0^2, \quad x_3, x_7 \geq 0,$$

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
*234567890123456789012345678901234567890123456789012345678901234567890						
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	$\geq 0$	unused	Zero cone (16.3).
QUAD	$\geq 1$	unused	Quadratic cone (16.4).
RQUAD	$\geq 2$	unused	Rotated quadratic cone (16.5).
PEXP	3	unused	Primal exponential cone (16.6).
PPOW	$\geq 2$	$\alpha$	Primal power cone (16.7).
DEXP	3	unused	Dual exponential cone (16.8).
DPOW	$\geq 2$	$\alpha$	Dual power cone (16.9).

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`.

## 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  $x_1, \dots, x_n$  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  $x_1, \dots, x_n$  defines a constraint of the form

$$2x_1x_2 \geq \sum_{i=3}^n x_i^2, \quad x_1, x_2 \geq 0.$$

- **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  $x_1, \dots, x_n$  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  $x_1, \dots, x_n$  defines a constraint of the form

$$\left(\frac{x_1}{\alpha}\right)^\alpha \left(\frac{x_2}{1-\alpha}\right)^{1-\alpha} \geq \sqrt{\sum_{j=3}^n x_j^2}, \quad x_1, x_2 \geq 0.$$

- **zero**: zero cone of  $n$  variables  $x_1, \dots, x_n$  defines a constraint of the form

$$x_1 = \dots = x_n = 0$$

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).
- **lv1** Defines the level of the item.
- **s1** 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**, **lv1**, **s1** and **su**. Items **s1** and **su** are not required for integer solutions.

A [con] section should always contain **sk**, **lv1**, **s1**, **su** and **y**.

An example of a solution section

```
[solution basic status=UNKNOWN]
[var x0] sk=LOW    lv1=5.0    [/var]
[var x1] sk=UPR    lv1=10.0   [/var]
[var x2] sk=SUPBAS lv1=2.0    s1=1.5 su=0.0 [/var]

[con c0] sk=LOW    lv1=3.0 y=0.0 [/con]
[con c0] sk=UPR    lv1=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:

<i>iparam.opf_write_sol_bas</i>	Include basic solution, if defined.
<i>iparam.opf_write_sol_itg</i>	Include integer solution, if defined.
<i>iparam.opf_write_sol_itr</i>	Include interior solution, if defined.
<i>iparam.opf_write_solutions</i>	Include solutions if they are defined. If this is off, no solutions are included.
<i>iparam.opf_write_header</i>	Include a small header with comments.
<i>iparam.opf_write_problem</i>	Include the problem itself — objective, constraints and bounds.
<i>iparam.opf_write_parameters</i>	Include all parameter settings.
<i>iparam.opf_write_hints</i>	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:

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

having the bounds

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

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)

```

[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

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

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)

```
[bounds]
[b] 0 <= * [/b]
[/bounds]
```

### Conic Quadratic Example `cqo1.opf`

Consider the example:

$$\begin{aligned}
&\text{minimize} && x_3 + x_4 + x_5 \\
&\text{subject to} && x_0 + x_1 + 2x_2 = 1, \\
& && x_0, x_1, x_2 \geq 0, \\
& && x_3 \geq \sqrt{x_0^2 + x_1^2}, \\
& && 2x_4x_5 \geq x_2^2.
\end{aligned}$$

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:

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

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) 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 problem structure is separated from the problem data, and the format moreover facilitates benchmarking of hotstart capability through sequences of changes.

### 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:

$$\begin{aligned} \min / \max \quad & g^{obj} \\ \text{s.t.} \quad & g_i \in \mathcal{K}_i, \quad i \in \mathcal{I}, \\ & G_i \in \mathcal{K}_i, \quad i \in \mathcal{I}^{PSD}, \\ & x_j \in \mathcal{K}_j, \quad j \in \mathcal{J}, \\ & \bar{X}_j \in \mathcal{K}_j, \quad j \in \mathcal{J}^{PSD}. \end{aligned} \tag{16.10}$$

- **Variables** are either scalar variables,  $x_j$  for  $j \in \mathcal{J}$ , or variables,  $\bar{X}_j$  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  $i \in \mathcal{I}^{PSD}$

$$\begin{aligned} g_i &= \sum_{j \in \mathcal{J}^{PSD}} \langle F_{ij}, X_j \rangle + \sum_{j \in \mathcal{J}} a_{ij} x_j + b_i, \\ G_i &= \sum_{j \in \mathcal{J}} x_j H_{ij} + D_i. \end{aligned}$$

- The **objective function** is a scalar-valued affine expression of the variables, either to be minimized or maximized. We refer to this expression as  $g^{obj}$

$$g^{obj} = \sum_{j \in \mathcal{J}^{PSD}} \langle F_j^{obj}, X_j \rangle + \sum_{j \in \mathcal{J}} a_j^{obj} x_j + b^{obj}.$$

CBF format can represent the following 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

$$\{x \in \mathbb{R}^n \mid x_j \geq 0 \text{ for } j = 1, \dots, n\}, \text{ for } n \geq 1.$$

- **Negative orthant** - A cone in the linear family defined by

$$\{x \in \mathbb{R}^n \mid x_j \leq 0 \text{ for } j = 1, \dots, n\}, \text{ for } n \geq 1.$$

- **Fixpoint zero** - A cone in the linear family defined by

$$\{x \in \mathbb{R}^n \mid x_j = 0 \text{ for } j = 1, \dots, n\}, \text{ for } n \geq 1.$$

- **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

$$\left\{ \begin{pmatrix} p \\ q \\ x \end{pmatrix} \in \mathbb{R} \times \mathbb{R} \times \mathbb{R}^{n-2}, 2pq \geq x^T x, p \geq 0, q \geq 0 \right\}, \text{ for } n \geq 3.$$

### 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.

#### Embedded hotstart-sequences

A sequence of problem instances, based on the same problem structure, is within a single file. This is facilitated via the **CHANGE** within the problem data information group, as a separator between the information items of each instance. The information items following a **CHANGE** keyword are appending to, or changing (e.g., setting coefficients back to their default value of zero), the problem data of the preceding instance.

The sequence is intended for benchmarking of hotstart capability, where the solvers can reuse their internal state and solution (subject to the achieved accuracy) as warmpoint for the succeeding instance. Whenever this feature is unsupported or undesired, the keyword **CHANGE** should be interpreted as the end of file.

### 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 separator 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,  $\{0, 1, \dots\}$ , 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

$$x \in \mathcal{K}_1^{n_1} \times \mathcal{K}_2^{n_2} \times \dots \times \mathcal{K}_k^{n_k}$$

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

$$g \in \mathcal{K}_1^{m_1} \times \mathcal{K}_2^{m_2} \times \dots \times \mathcal{K}_k^{m_k}$$

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 minimum sizes are given as follows.

Table 16.3: Cones available in the CBF format

Name	CBF keyword	Cone family
Free domain	F	linear
Positive orthant	L+	linear
Negative orthant	L-	linear
Fixpoint zero	L=	linear
Quadratic cone	Q	second-order
Rotated quadratic cone	QR	second-order

### 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.

#### OBJSENSE

*Description:* Define the objective sense.

HEADER: None

BODY: One line formatted as:

STR

having MIN indicates minimize, and MAX indicates maximize. Capital 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.

## CHANGE

Start of a new instance specification based on changes to the previous. Can be interpreted as the end of file when the hotstart-sequence is unsupported or undesired.

**BODY:** None

**Header:** None

## 16.4.5 CBF Format Examples

### Minimal Working Example

The conic optimization problem (16.11), has three variables in a quadratic cone - first one is integer - and an affine expression in domain 0 (equality constraint).

$$\begin{aligned} & \text{minimize} && 5.1 x_0 \\ & \text{subject to} && 6.2 x_1 + 7.3 x_2 - 8.4 \in \{0\} \\ & && x \in \mathcal{Q}^3, x_0 \in \mathbb{Z}. \end{aligned} \tag{16.11}$$

Its formulation in the Conic Benchmark Format begins with the version of the CBF format used, to safeguard against later revisions.

VER
1

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.

```
OBJCOORD
```

```
1
0 5.1
```

```
ACCOORD
```

```
2
0 1 6.2
0 2 7.3
```

```
BCCOORD
```

```
1
0 -8.4
```

This concludes the example.

### Mixing Linear, Second-order and Semidefinite Cones

The conic optimization problem (16.12), has a semidefinite cone, a quadratic cone over unordered subindices, and two equality constraints.

$$\begin{aligned} & \text{minimize} && \left\langle \begin{bmatrix} 2 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 2 \end{bmatrix}, X_1 \right\rangle + x_1 \\ & \text{subject to} && \left\langle \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, X_1 \right\rangle + x_1 &= 1.0, \\ & && \left\langle \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, X_1 \right\rangle + x_0 + x_2 &= 0.5, \\ & && x_1 \geq \sqrt{x_0^2 + x_2^2}, \\ & && X_1 \succeq \mathbf{0}. \end{aligned} \tag{16.12}$$

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 1.
VER
1

# 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
```

(continues on next page)

```

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
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

```

(continues on next page)

(continued from previous page)

```
# 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 in.

$$\begin{aligned} \text{minimize} \quad & \left\langle \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, X_1 \right\rangle + x_1 + x_2 + 1 \\ \text{subject to} \quad & \left\langle \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, X_1 \right\rangle - x_1 - x_2 \geq 0.0, \\ & x_1 \begin{bmatrix} 0 & 1 \\ 1 & 3 \end{bmatrix} + x_2 \begin{bmatrix} 3 & 1 \\ 1 & 0 \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \preceq \mathbf{0}, \\ & X_1 \succeq \mathbf{0}. \end{aligned} \tag{16.13}$$

Its formulation in the CBF format is written in what follows

```
# File written using this version of the Conic Benchmark Format:
#      | Version 1.
VER
1

# 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
```

(continues on next page)

```

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:
#   |  $a^{\{obj\}}[0] = 1.0$ 
#   |  $a^{\{obj\}}[1] = 1.0$ 
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:
#   |  $a[0,0] = -1.0$ 
#   |  $a[0,1] = -1.0$ 
ACCOORD
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:
#   |  $D[0][0,0] = -1.0$ 
#   |  $D[0][1,1] = -1.0$ 
DCCOORD
2
0 0 0 -1.0
0 1 1 -1.0

```

## Optimization Over a Sequence of Objectives

The linear optimization problem (16.14), is defined for a sequence of objectives such that hotstarting from one to the next might be advantages.

$$\begin{aligned} & \text{maximize}_k && g_k^{obj} \\ & \text{subject to} && 50x_0 + 31 \leq 250, \\ & && 3x_0 - 2x_1 \geq -4, \\ & && x \in \mathbb{R}_+^2, \end{aligned} \tag{16.14}$$

given,

1.  $g_0^{obj} = x_0 + 0.64x_1$ .
2.  $g_1^{obj} = 1.11x_0 + 0.76x_1$ .
3.  $g_2^{obj} = 1.11x_0 + 0.85x_1$ .

Its formulation in the CBF format is reported in Listing 16.5.

Listing 16.5: Problem (16.14) in CBF format.

```
# File written using this version of the Conic Benchmark Format:
#   | Version 1.
VER
1

# The sense of the objective is:
#   | Maximize.
OBJSENSE
MAX

# Two scalar variables in this one conic domain:
#   | Two are nonnegative.
VAR
2 1
L+ 2

# Two scalar constraints with affine expressions in these two conic domains:
#   | One is in the nonpositive domain.
#   | One is in the nonnegative domain.
CON
2 2
L- 1
L+ 1

# Two coordinates in a^{obj}_j coefficients:
#   | a^{obj}[0] = 1.0
#   | a^{obj}[1] = 0.64
OBJACoord
2
0 1.0
1 0.64

# Four coordinates in a_ij coefficients:
#   | a[0,0] = 50.0
#   | a[1,0] = 3.0
#   | and more...
ACoord
4
```

(continues on next page)

```

0 0 50.0
1 0 3.0
0 1 31.0
1 1 -2.0

# Two coordinates in b_i coefficients:
#     | b[0] = -250.0
#     | b[1] = 4.0
BCOORD
2
0 -250.0
1 4.0

# New problem instance defined in terms of changes.
CHANGE

# Two coordinate changes in a^{obj}_j coefficients. Now it is:
#     | a^{obj}[0] = 1.11
#     | a^{obj}[1] = 0.76
OBJACOORD
2
0 1.11
1 0.76

# New problem instance defined in terms of changes.
CHANGE

# One coordinate change in a^{obj}_j coefficients. Now it is:
#     | a^{obj}[0] = 1.11
#     | a^{obj}[1] = 0.85
OBJACOORD
1
1 0.85

```

## 16.5 The PTF Format

The PTF format is a new human-readable, natural text format. Its features and structure are similar to the *OPF* format, with the difference that the PTF format **does** support semidefinite terms.

### 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 than 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

```

(continues on next page)

```

    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 ) *
TERM: LINEAR_TERM | MATRIX_TERM

```

A linear term

```

LINEAR_TERM: FLOAT? NAME

```

A matrix term

```

MATRIX_TERM: "<" FLOAT? NAME ( [+]? 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 matrixes. The actual matrixes 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 separate 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. If they contain one value the constraint is an equality.

For example:

```
Constraints
'c1' [0;10] x1 + x2 + x3
[0] 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:

- SOC(N) Second order cone of dimension N
- RSOC(N) Rotated second order cone of dimension N
- PSD(N) Symmetric positive semidefinite cone of dimension N. This contains  $N*(N+1)/2$  elements.
- PEXP Primal exponential cone of dimension 3
- DEXP Dual exponential cone of dimension 3
- PPOW(N,P) Primal power cone of dimension N with parameter P
- DPOW(N,P) Dual power cone of dimension N with parameter P
- ZERO(N) The zero-cone of dimension N.

```
"Constraints"
NAME? "[" DOMAIN "]" EXPR_LIST
```

For example:

```
Constraints
'K1' [SOC(3)] x1 + x2 ; x2 + x3 ; x3 + x1
'K2' [RSOC(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 "[" DOMAIN "]" NAMES

For example, a linear variable
```

```
Variables
x1 [0;Inf]
```

As with constraints, members of a conic domain can be listed either inline or in a subsection:

```
Variables
k1 [SOC(3)] x1 ; x2 ; x3
k2 [RSOC(3)]
    x1
    x2
    x3
```

### 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
  "Variables"
    # Linear variable status: level, slx, sux
    NAME "[" STATUS "]" FLOAT (FLOAT FLOAT)?
    # Conic variable status: level, snx
    NAME
      "[" STATUS "]" FLOAT FLOAT?
    ...
  "Constraints"
    # Linear variable status: level, slx, sux
    NAME "[" STATUS "]" FLOAT (FLOAT FLOAT)?
    # Conic variable status: level, snx
    NAME
      "[" STATUS "]" FLOAT 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

## 16.6 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 in valid JSON format.

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.

The official JSON website <http://www.json.org> provides plenty of information along with the format definition.

**MOSEK** defines two JSON-like formats:

- `jtask`
- `jsol`

Despite being text-based human-readable formats, *jtask* and *jsol* files will include no indentation and no new-lines, in order to keep the files as compact as possible. We therefore strongly advise to use JSON viewer tools to inspect *jtask* and *jsol* files.

### 16.7.1 *jtask* format

It stores a problem instance. The *jtask* format contains the same information as a *task format*. Even though a *jtask* file is human-readable, we do not recommend users to create it by hand, but to rely on MOSEK.

### 16.7.2 *jsol* format

It stores a problem solution. The *jsol* format contains all solutions and information items.

You can write a *jsol* file using `Task.writejsonsol`. You **can not** read a *jsol* file into MOSEK.

### 16.7.3 A *jtask* example

In Listing 16.6 we present a file in the *jtask* format that corresponds to the sample problem from 1o1.1p. The listing has been formatted for readability.

Listing 16.6: A formatted *jtask* file for the 1o1.1p example.

```
{
  "$schema": "http://mosek.com/json/schema#",
  "Task/INFO": {
    "taskname": "1o1",
    "numvar": 4,
    "numcon": 3,
    "numcone": 0,
    "numbarvar": 0,
    "numanz": 9,
    "numsymmat": 0,
    "mosekver": [
      8,
      0,
      0,
      9
    ]
  },
  "Task/data": {
    "var": {
      "name": [
        "x1",
        "x2",
        "x3",
        "x4"
      ],
      "bk": [
        "lo",
        "ra",
        "lo",
        "lo"
      ],
      "b1": [
        0.0,
        0.0,
        0.0,
        0.0
      ],
      "bu": [
        1e+30,
        1e+1,

```

(continues on next page)

```

        1e+30,
        1e+30
    ],
    "type": [
        "cont",
        "cont",
        "cont",
        "cont"
    ]
},
"con": {
    "name": [
        "c1",
        "c2",
        "c3"
    ],
    "bk": [
        "fx",
        "lo",
        "up"
    ],
    "bl": [
        3e+1,
        1.5e+1,
        -1e+30
    ],
    "bu": [
        3e+1,
        1e+30,
        2.5e+1
    ]
},
"objective": {
    "sense": "max",
    "name": "obj",
    "c": {
        "subj": [
            0,
            1,
            2,
            3
        ],
        "val": [
            3e+0,
            1e+0,
            5e+0,
            1e+0
        ]
    },
    "cfix": 0.0
},
"A": {
    "subi": [
        0,
        0,
        0,

```

```

        1,
        1,
        1,
        1,
        2,
        2
    ],
    "subj": [
        0,
        1,
        2,
        0,
        1,
        2,
        3,
        1,
        3
    ],
    "val": [
        3e+0,
        1e+0,
        2e+0,
        2e+0,
        1e+0,
        3e+0,
        1e+0,
        2e+0,
        3e+0
    ]
}
},
"Task/parameters": {
    "iparam": {
        "ANA_SOL_BASIS": "ON",
        "ANA_SOL_PRINT_VIOLATED": "OFF",
        "AUTO_SORT_A_BEFORE_OPT": "OFF",
        "AUTO_UPDATE_SOL_INFO": "OFF",
        "BASIS_SOLVE_USE_PLUS_ONE": "OFF",
        "BI_CLEAN_OPTIMIZER": "OPTIMIZER_FREE",
        "BI_IGNORE_MAX_ITER": "OFF",
        "BI_IGNORE_NUM_ERROR": "OFF",
        "BI_MAX_ITERATIONS": 1000000,
        "CACHE_LICENSE": "ON",
        "CHECK_CONVEXITY": "CHECK_CONVEXITY_FULL",
        "COMPRESS_STATFILE": "ON",
        "CONCURRENT_NUM_OPTIMIZERS": 2,
        "CONCURRENT_PRIORITY_DUAL_SIMPLEX": 2,
        "CONCURRENT_PRIORITY_FREE_SIMPLEX": 3,
        "CONCURRENT_PRIORITY_INTPNT": 4,
        "CONCURRENT_PRIORITY_PRIMAL_SIMPLEX": 1,
        "FEASREPAIR_OPTIMIZE": "FEASREPAIR_OPTIMIZE_NONE",
        "INFEAS_GENERIC_NAMES": "OFF",
        "INFEAS_PREFER_PRIMAL": "ON",
        "INFEAS_REPORT_AUTO": "OFF",
        "INFEAS_REPORT_LEVEL": 1,
        "INTPNT_BASIS": "BI_ALWAYS",
    }
}

```

```

"INTPNT_DIFF_STEP": "ON",
"INTPNT_FACTOR_DEBUG_LVL": 0,
"INTPNT_FACTOR_METHOD": 0,
"INTPNT_HOTSTART": "INTPNT_HOTSTART_NONE",
"INTPNT_MAX_ITERATIONS": 400,
"INTPNT_MAX_NUM_COR": -1,
"INTPNT_MAX_NUM_REFINEMENT_STEPS": -1,
"INTPNT_OFF_COL_TRH": 40,
"INTPNT_ORDER_METHOD": "ORDER_METHOD_FREE",
"INTPNT_REGULARIZATION_USE": "ON",
"INTPNT_SCALING": "SCALING_FREE",
"INTPNT_SOLVE_FORM": "SOLVE_FREE",
"INTPNT_STARTING_POINT": "STARTING_POINT_FREE",
"LIC_TRH_EXPIRY_WRN": 7,
"LICENSE_DEBUG": "OFF",
"LICENSE_PAUSE_TIME": 0,
"LICENSE_SUPPRESS_EXPIRE_WRNS": "OFF",
"LICENSE_WAIT": "OFF",
"LOG": 10,
"LOG_ANA_PRO": 1,
"LOG_BI": 4,
"LOG_BI_FREQ": 2500,
"LOG_CHECK_CONVEXITY": 0,
"LOG_CONCURRENT": 1,
"LOG_CUT_SECOND_OPT": 1,
"LOG_EXPAND": 0,
"LOG_FACTOR": 1,
"LOG_FEAS_REPAIR": 1,
"LOG_FILE": 1,
"LOG_HEAD": 1,
"LOG_INFEAS_ANA": 1,
"LOG_INTPNT": 4,
"LOG_MIO": 4,
"LOG_MIO_FREQ": 1000,
"LOG_OPTIMIZER": 1,
"LOG_ORDER": 1,
"LOG_PRESOLVE": 1,
"LOG_RESPONSE": 0,
"LOG_SENSITIVITY": 1,
"LOG_SENSITIVITY_OPT": 0,
"LOG_SIM": 4,
"LOG_SIM_FREQ": 1000,
"LOG_SIM_MINOR": 1,
"LOG_STORAGE": 1,
"MAX_NUM_WARNINGS": 10,
"MIO_BRANCH_DIR": "BRANCH_DIR_FREE",
"MIO_CONSTRUCT_SOL": "OFF",
"MIO_CUT_CLIQUE": "ON",
"MIO_CUT_CMIR": "ON",
"MIO_CUT_GMI": "ON",
"MIO_CUT_KNAPSACK_COVER": "OFF",
"MIO_HEURISTIC_LEVEL": -1,
"MIO_MAX_NUM_BRANCHES": -1,
"MIO_MAX_NUM_RELAXS": -1,
"MIO_MAX_NUM_SOLUTIONS": -1,
"MIO_MODE": "MIO_MODE_SATISFIED",

```

```

"MIO_MT_USER_CB": "ON",
"MIO_NODE_OPTIMIZER": "OPTIMIZER_FREE",
"MIO_NODE_SELECTION": "MIO_NODE_SELECTION_FREE",
"MIO_PERSPECTIVE_REFORMULATE": "ON",
"MIO_PROBING_LEVEL": -1,
"MIO_RINS_MAX_NODES": -1,
"MIO_ROOT_OPTIMIZER": "OPTIMIZER_FREE",
"MIO_ROOT_REPEAT_PRESOLVE_LEVEL": -1,
"MT_SPINCOUNT": 0,
"NUM_THREADS": 0,
"OPF_MAX_TERMS_PER_LINE": 5,
"OPF_WRITE_HEADER": "ON",
"OPF_WRITE_HINTS": "ON",
"OPF_WRITE_PARAMETERS": "OFF",
"OPF_WRITE_PROBLEM": "ON",
"OPF_WRITE_SOL_BAS": "ON",
"OPF_WRITE_SOL_ITG": "ON",
"OPF_WRITE_SOL_ITR": "ON",
"OPF_WRITE_SOLUTIONS": "OFF",
"OPTIMIZER": "OPTIMIZER_FREE",
"PARAM_READ_CASE_NAME": "ON",
"PARAM_READ_IGN_ERROR": "OFF",
"PRESOLVE_ELIMINATOR_MAX_FILL": -1,
"PRESOLVE_ELIMINATOR_MAX_NUM_TRIES": -1,
"PRESOLVE_LEVEL": -1,
"PRESOLVE_LINDEP_ABS_WORK_TRH": 100,
"PRESOLVE_LINDEP_REL_WORK_TRH": 100,
"PRESOLVE_LINDEP_USE": "ON",
"PRESOLVE_MAX_NUM_REDUCTIONS": -1,
"PRESOLVE_USE": "PRESOLVE_MODE_FREE",
"PRIMAL_REPAIR_OPTIMIZER": "OPTIMIZER_FREE",
"QO_SEPARABLE_REFORMULATION": "OFF",
"READ_DATA_COMPRESSED": "COMPRESS_FREE",
"READ_DATA_FORMAT": "DATA_FORMAT_EXTENSION",
"READ_DEBUG": "OFF",
"READ_KEEP_FREE_CON": "OFF",
"READ_LP_DROP_NEW_VARS_IN_BOU": "OFF",
"READ_LP_QUOTED_NAMES": "ON",
"READ_MPS_FORMAT": "MPS_FORMAT_FREE",
"READ_MPS_WIDTH": 1024,
"READ_TASK_IGNORE_PARAM": "OFF",
"SENSITIVITY_ALL": "OFF",
"SENSITIVITY_OPTIMIZER": "OPTIMIZER_FREE_SIMPLEX",
"SENSITIVITY_TYPE": "SENSITIVITY_TYPE_BASIS",
"SIM_BASIS_FACTOR_USE": "ON",
"SIM_DEGEN": "SIM_DEGEN_FREE",
"SIM_DUAL_CRASH": 90,
"SIM_DUAL_PHASEONE_METHOD": 0,
"SIM_DUAL_RESTRICT_SELECTION": 50,
"SIM_DUAL_SELECTION": "SIM_SELECTION_FREE",
"SIM_EXPLOIT_DUPVEC": "SIM_EXPLOIT_DUPVEC_OFF",
"SIM_HOTSTART": "SIM_HOTSTART_FREE",
"SIM_HOTSTART_LU": "ON",
"SIM_INTEGER": 0,
"SIM_MAX_ITERATIONS": 10000000,
"SIM_MAX_NUM_SETBACKS": 250,

```

```

"SIM_NON_SINGULAR": "ON",
"SIM_PRIMAL_CRASH": 90,
"SIM_PRIMAL_PHASEONE_METHOD": 0,
"SIM_PRIMAL_RESTRICT_SELECTION": 50,
"SIM_PRIMAL_SELECTION": "SIM_SELECTION_FREE",
"SIM_REFACTOR_FREQ": 0,
"SIM_REFORMULATION": "SIM_REFORMULATION_OFF",
"SIM_SAVE_LU": "OFF",
"SIM_SCALING": "SCALING_FREE",
"SIM_SCALING_METHOD": "SCALING_METHOD_POW2",
"SIM_SOLVE_FORM": "SOLVE_FREE",
"SIM_STABILITY_PRIORITY": 50,
"SIM_SWITCH_OPTIMIZER": "OFF",
"SOL_FILTER_KEEP_BASIC": "OFF",
"SOL_FILTER_KEEP_RANGED": "OFF",
"SOL_READ_NAME_WIDTH": -1,
"SOL_READ_WIDTH": 1024,
"SOLUTION_CALLBACK": "OFF",
"TIMING_LEVEL": 1,
"WRITE_BAS_CONSTRAINTS": "ON",
"WRITE_BAS_HEAD": "ON",
"WRITE_BAS_VARIABLES": "ON",
"WRITE_DATA_COMPRESSED": 0,
"WRITE_DATA_FORMAT": "DATA_FORMAT_EXTENSION",
"WRITE_DATA_PARAM": "OFF",
"WRITE_FREE_CON": "OFF",
"WRITE_GENERIC_NAMES": "OFF",
"WRITE_GENERIC_NAMES_IO": 1,
"WRITE_IGNORE_INCOMPATIBLE_CONIC_ITEMS": "OFF",
"WRITE_IGNORE_INCOMPATIBLE_ITEMS": "OFF",
"WRITE_IGNORE_INCOMPATIBLE_NL_ITEMS": "OFF",
"WRITE_IGNORE_INCOMPATIBLE_PSD_ITEMS": "OFF",
"WRITE_INT_CONSTRAINTS": "ON",
"WRITE_INT_HEAD": "ON",
"WRITE_INT_VARIABLES": "ON",
"WRITE_LP_FULL_OBJ": "ON",
"WRITE_LP_LINE_WIDTH": 80,
"WRITE_LP_QUOTED_NAMES": "ON",
"WRITE_LP_STRICT_FORMAT": "OFF",
"WRITE_LP_TERMS_PER_LINE": 10,
"WRITE_MPS_FORMAT": "MPS_FORMAT_FREE",
"WRITE_MPS_INT": "ON",
"WRITE_PRECISION": 15,
"WRITE_SOL_BARVARIABLES": "ON",
"WRITE_SOL_CONSTRAINTS": "ON",
"WRITE_SOL_HEAD": "ON",
"WRITE_SOL_IGNORE_INVALID_NAMES": "OFF",
"WRITE_SOL_VARIABLES": "ON",
"WRITE_TASK_INC_SOL": "ON",
"WRITE_XML_MODE": "WRITE_XML_MODE_ROW"
},
"dparam": {
  "ANA_SOL_INFEAS_TOL": 1e-6,
  "BASIS_REL_TOL_S": 1e-12,
  "BASIS_TOL_S": 1e-6,
  "BASIS_TOL_X": 1e-6,

```

```

"CHECK_CONVEXITY_REL_TOL":1e-10,
"DATA_TOL_AIJ":1e-12,
"DATA_TOL_AIJ_HUGE":1e+20,
"DATA_TOL_AIJ_LARGE":1e+10,
"DATA_TOL_BOUND_INF":1e+16,
"DATA_TOL_BOUND_WRN":1e+8,
"DATA_TOL_C_HUGE":1e+16,
"DATA_TOL_CJ_LARGE":1e+8,
"DATA_TOL_QIJ":1e-16,
"DATA_TOL_X":1e-8,
"FEASREPAIR_TOL":1e-10,
"INTPNT_CO_TOL_DFEAS":1e-8,
"INTPNT_CO_TOL_INFEAS":1e-10,
"INTPNT_CO_TOL_MU_RED":1e-8,
"INTPNT_CO_TOL_NEAR_REL":1e+3,
"INTPNT_CO_TOL_PFEAS":1e-8,
"INTPNT_CO_TOL_REL_GAP":1e-7,
"INTPNT_NL_MERIT_BAL":1e-4,
"INTPNT_NL_TOL_DFEAS":1e-8,
"INTPNT_NL_TOL_MU_RED":1e-12,
"INTPNT_NL_TOL_NEAR_REL":1e+3,
"INTPNT_NL_TOL_PFEAS":1e-8,
"INTPNT_NL_TOL_REL_GAP":1e-6,
"INTPNT_NL_TOL_REL_STEP":9.95e-1,
"INTPNT_QO_TOL_DFEAS":1e-8,
"INTPNT_QO_TOL_INFEAS":1e-10,
"INTPNT_QO_TOL_MU_RED":1e-8,
"INTPNT_QO_TOL_NEAR_REL":1e+3,
"INTPNT_QO_TOL_PFEAS":1e-8,
"INTPNT_QO_TOL_REL_GAP":1e-8,
"INTPNT_TOL_DFEAS":1e-8,
"INTPNT_TOL_DSAFE":1e+0,
"INTPNT_TOL_INFEAS":1e-10,
"INTPNT_TOL_MU_RED":1e-16,
"INTPNT_TOL_PATH":1e-8,
"INTPNT_TOL_PFEAS":1e-8,
"INTPNT_TOL_PSAFE":1e+0,
"INTPNT_TOL_REL_GAP":1e-8,
"INTPNT_TOL_REL_STEP":9.999e-1,
"INTPNT_TOL_STEP_SIZE":1e-6,
"LOWER_OBJ_CUT":-1e+30,
"LOWER_OBJ_CUT_FINITE_TRH":-5e+29,
"MIO_DISABLE_TERM_TIME":-1e+0,
"MIO_MAX_TIME":-1e+0,
"MIO_MAX_TIME_APRX_OPT":6e+1,
"MIO_NEAR_TOL_ABS_GAP":0.0,
"MIO_NEAR_TOL_REL_GAP":1e-3,
"MIO_REL_GAP_CONST":1e-10,
"MIO_TOL_ABS_GAP":0.0,
"MIO_TOL_ABS_RELAX_INT":1e-5,
"MIO_TOL_FEAS":1e-6,
"MIO_TOL_REL_DUAL_BOUND_IMPROVEMENT":0.0,
"MIO_TOL_REL_GAP":1e-4,
"MIO_TOL_X":1e-6,
"OPTIMIZER_MAX_TIME":-1e+0,
"PRESOLVE_TOL_ABS_LINDEP":1e-6,

```

```

        "PRESOLVE_TOL_AIJ":1e-12,
        "PRESOLVE_TOL_REL_LINDEP":1e-10,
        "PRESOLVE_TOL_S":1e-8,
        "PRESOLVE_TOL_X":1e-8,
        "QCQO_REFORMULATE_REL_DROP_TOL":1e-15,
        "SEMIDEFINITE_TOL_APPROX":1e-10,
        "SIM_LU_TOL_REL_PIV":1e-2,
        "SIMPLEX_ABS_TOL_PIV":1e-7,
        "UPPER_OBJ_CUT":1e+30,
        "UPPER_OBJ_CUT_FINITE_TRH":5e+29
    },
    "sparam":{
        "BAS_SOL_FILE_NAME":"",
        "DATA_FILE_NAME":"examples/tools/data/lo1.mps",
        "DEBUG_FILE_NAME":"",
        "INT_SOL_FILE_NAME":"",
        "ITR_SOL_FILE_NAME":"",
        "MIO_DEBUG_STRING":"",
        "PARAM_COMMENT_SIGN":"%",
        "PARAM_READ_FILE_NAME":"",
        "PARAM_WRITE_FILE_NAME":"",
        "READ_MPS_BOU_NAME":"",
        "READ_MPS_OBJ_NAME":"",
        "READ_MPS_RAN_NAME":"",
        "READ_MPS_RHS_NAME":"",
        "SENSITIVITY_FILE_NAME":"",
        "SENSITIVITY_RES_FILE_NAME":"",
        "SOL_FILTER_XC_LOW":"",
        "SOL_FILTER_XC_UPR":"",
        "SOL_FILTER_XX_LOW":"",
        "SOL_FILTER_XX_UPR":"",
        "STAT_FILE_NAME":"",
        "STAT_KEY":"",
        "STAT_NAME":"",
        "WRITE_LP_GEN_VAR_NAME":"XMSKGEN"
    }
}
}

```

## 16.8 The Solution File Format

MOSEK provides several solution files depending on the problem type and the optimizer used:

- *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 contains integer constrained variables.

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>

(continues on next page)

(continued from previous page)

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
?	<name>	??	<a value>	<a value>	<a value>	<a value>
VARIABLES						
INDEX	NAME	AT	ACTIVITY	LOWER LIMIT	UPPER LIMIT	DUAL LOWER
?	<name>	??	<a value>	<a value>	<a value>	<a value>
↪ CONIC DUAL						
?	<name>	??	<a value>	<a value>	<a value>	<a value>
↪ <a value>						

In the example the fields ? and <> will be filled with problem and solution specific information. As can be observed a solution report consists of three sections, i.e.

- **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** For each constraint  $i$  of the form

$$l_i^c \leq \sum_{j=1}^n a_{ij}x_j \leq u_i^c, \quad (16.15)$$

the following information is listed:

- **INDEX**: A sequential index assigned to the constraint by **MOSEK**
- **NAME**: The name of the constraint assigned by the user.
- **AT**: The status of the constraint. In Table 16.4 the possible values of the status keys and their interpretation are shown.

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.

- **ACTIVITY**: the quantity  $\sum_{j=1}^n a_{ij}x_j^*$ , where  $x^*$  is the value of the primal solution.
- **LOWER LIMIT**: the quantity  $l_i^c$  (see (16.15).)
- **UPPER LIMIT**: the quantity  $u_i^c$  (see (16.15).)
- **DUAL LOWER**: the dual multiplier corresponding to the lower limit on the constraint.
- **DUAL UPPER**: the dual multiplier corresponding to the upper limit on the constraint.
- **VARIABLES** The last section of the solution report lists information about the variables. This information has a similar interpretation as for the constraints. However, the column with the header **CONIC DUAL** is included for problems having one or more conic constraints. This column shows the dual variables corresponding to the conic constraints.

**Example: lo1.sol**

In Listing 16.7 we show the solution file for the lo1.opf problem.

Listing 16.7: An example of .sol file.

NAME	:				
PROBLEM STATUS	:	PRIMAL_AND_DUAL_FEASIBLE			
SOLUTION STATUS	:	OPTIMAL			
OBJECTIVE NAME	:	obj			
PRIMAL OBJECTIVE	:	8.33333333e+01			
DUAL OBJECTIVE	:	8.33333332e+01			
CONSTRAINTS					
INDEX	NAME	AT	ACTIVITY	LOWER LIMIT	UPPER LIMIT
	DUAL LOWER		DUAL UPPER		
0	c1	EQ	3.00000000000000e+01	3.00000000e+01	3.
	00000000e+01		-0.00000000000000e+00	-2.49999999741653e+00	
1	c2	SB	5.33333333049187e+01	1.50000000e+01	NONE
	2.09159033069640e-10		-0.00000000000000e+00		
2	c3	UL	2.49999999842049e+01	NONE	2.
	50000000e+01		-0.00000000000000e+00	-3.33333332895108e-01	
VARIABLES					
INDEX	NAME	AT	ACTIVITY	LOWER LIMIT	UPPER LIMIT
	DUAL LOWER		DUAL UPPER		
0	x1	LL	1.67020427038537e-09	0.00000000e+00	NONE
	-4.49999999528054e+00		-0.00000000000000e+00		
1	x2	LL	2.93510446211883e-09	0.00000000e+00	1.
	00000000e+01		-2.16666666494915e+00	6.20868657679896e-10	
2	x3	SB	1.49999999899424e+01	0.00000000e+00	NONE
	-8.79123177245553e-10		-0.00000000000000e+00		
3	x4	SB	8.33333332273115e+00	0.00000000e+00	NONE
	-1.69795978848200e-09		-0.00000000000000e+00		

# Chapter 17

## List of examples

List of examples shipped in the distribution of Optimizer API for .NET:

Table 17.1: List of distributed examples

File	Description
blas_lapack.cs	Demonstrates the <b>MOSEK</b> interface to BLAS/LAPACK linear algebra routines
callback.cs	An example of data/progress callback
ceo1.cs	A simple conic exponential problem
concurrent1.cs	Implementation of a concurrent optimizer for linear and mixed-integer problems
cqo1.cs	A simple conic quadratic problem
feasrepairex1.cs	A simple example of how to repair an infeasible problem
gp1.cs	A simple geometric program (GP) in conic form
lo1.cs	A simple linear problem
lo1.vb	A simple linear problem
lo2.cs	A simple linear problem
logistic.cs	Implements logistic regression and simple log-sum-exp (CEO)
mico1.cs	A simple mixed-integer conic problem
milol.cs	A simple mixed-integer linear problem
miointsol.cs	A simple mixed-integer linear problem with an initial guess
modelLib.cs	Library of implementations of basic functions
opt_server_async.cs	Uses <b>MOSEK</b> OptServer to solve an optimization problem asynchronously
opt_server_sync.cs	Uses <b>MOSEK</b> OptServer to solve an optimization problem synchronously
parallel.cs	Demonstrates parallel optimization
parameters.cs	Shows how to set optimizer parameters and read information items
portfolio_1_basic.cs	Portfolio optimization - basic Markowitz model
portfolio_2_frontier.cs	Portfolio optimization - efficient frontier
portfolio_3_impact.cs	Portfolio optimization - market impact costs
portfolio_4_transaction.cs	Portfolio optimization - transaction costs
portfolio_5_cardinality.cs	Portfolio optimization - cardinality constraints
pow1.cs	A simple power cone problem
qcqo1.cs	A simple quadratically constrained quadratic problem
qo1.cs	A simple quadratic problem
reoptimization.cs	Demonstrate how to modify and re-optimize a linear problem

continues on next page

Table 17.1 – continued from previous page

File	Description
<code>response.cs</code>	Demonstrates proper response handling
<code>sdo1.cs</code>	A simple semidefinite problem with one matrix variable and a quadratic cone
<code>sdo2.cs</code>	A simple semidefinite problem with two matrix variables
<code>sensitivity.cs</code>	Sensitivity analysis performed on a small linear problem
<code>simple.cs</code>	A simple I/O example: read problem from a file, solve and write solutions
<code>solutionquality.cs</code>	Demonstrates how to examine the quality of a solution
<code>solvebasis.cs</code>	Demonstrates solving a linear system with the basis matrix
<code>solvelinear.cs</code>	Demonstrates solving a general linear system
<code>sparsecholesky.cs</code>	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 .NET in version 9.3 compared to version 8. See the [release notes](#) for general changes and new features of the **MOSEK** Optimization Suite.

### 18.1 Backwards compatibility

- **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.
- All functions using the enum `accmode` were removed. Use corresponding separate functions for manipulating variables and constraints. For example, instead of

```
task.putbound(accmode.var, ...);  
task.putbound(accmode.con, ...);
```

use

```
task.putvarbound(...);  
task.putconbound(...);
```

and so on.

- Removed all Near problem and solution statuses i.e. `solsta.near_optimal`, `solsta.near_prim_infeas_cer`, etc. See [Sec. 13.3.3](#).
- All functions related to the general nonlinear optimizer and `Scopt` have been removed. See [Sec. 15.11](#).

### 18.2 Functions

#### Added

- *`Env.setupthreads`*
- *`Task.appendsparsesymmatlist`*
- *`Task.generateconenames`*
- *`Task.generateconnames`*
- *`Task.generatevarnames`*
- *`Task.getacolslice`*

- *Task.getacolslicenumz*
- *Task.getarowslice*
- *Task.getarowslicenumz*
- *Task.getatruncatetol*
- *Task.getbarsslice*
- *Task.getbarxslice*
- *Task.getclist*
- *Task.getskn*
- *Task.putatruncatetol*
- *Task.putbaraijlist*
- *Task.putbararowlist*
- *Task.putconboundlistconst*
- *Task.putconboundsliceconst*
- *Task.putconsolutioni*
- *Task.putoptserverhost*
- *Task.putvarboundlistconst*
- *Task.putvarboundsliceconst*
- *Task.putvarsolutionj*
- *Task.readjsonstring*
- *Task.readlpstring*
- *Task.readopfstring*
- *Task.readptfstring*

#### Removed

- *Task.checkconvexity*
- *Task.chgbound*
- *Task.getaslice*
- *Task.getaslicenumz*
- *Task.getbound*
- *Task.getboundslice*
- *Task.getsolutioni*
- *Task.printdata*
- *Task.putbound*
- *Task.putboundlist*
- *Task.putboundslice*
- *Task.putsolutioni*

## 18.3 Parameters

### Added

- *iparam.intpnt\_order\_gp\_num\_seeds*
- *iparam.intpnt\_purify*
- *iparam.log\_include\_summary*
- *iparam.log\_local\_info*
- *iparam.mio\_conic\_outer\_approximation*
- *iparam.mio\_feaspump\_level*
- *iparam.mio\_max\_num\_root\_cut\_rounds*
- *iparam.mio\_propagate\_objective\_constraint*
- *iparam.mio\_seed*
- *iparam.opf\_write\_line\_length*
- *iparam.presolve\_max\_num\_pass*
- *iparam.ptf\_write\_transform*
- *iparam.sim\_seed*
- *iparam.write\_compression*

### Removed

- *dparam.data\_tol\_aij*
- *dparam.intpnt\_nl\_merit\_bal*
- *dparam.intpnt\_nl\_tol\_dfeas*
- *dparam.intpnt\_nl\_tol\_mu\_red*
- *dparam.intpnt\_nl\_tol\_near\_rel*
- *dparam.intpnt\_nl\_tol\_pfeas*
- *dparam.intpnt\_nl\_tol\_rel\_gap*
- *dparam.intpnt\_nl\_tol\_rel\_step*
- *dparam.mio\_disable\_term\_time*
- *dparam.mio\_near\_tol\_abs\_gap*
- *dparam.mio\_near\_tol\_rel\_gap*
- *iparam.mio\_construct\_sol*
- *iparam.mio\_mt\_user\_cb*
- *iparam.opf\_max\_terms\_per\_line*
- *iparam.read\_data\_compressed*
- *iparam.read\_data\_format*
- *iparam.write\_data\_compressed*
- *iparam.write\_data\_format*

## 18.4 Constants

### Added

- *compresstype.zstd*
- *conetype.dexp*
- *conetype.dpow*
- *conetype.pexp*
- *conetype.ppow*
- *conetype.zero*
- *dataformat.ptf*
- *infitem.mio\_numbin*
- *infitem.mio\_numbinconevar*
- *infitem.mio\_numcone*
- *infitem.mio\_numconevar*
- *infitem.mio\_numcont*
- *infitem.mio\_numcontconevar*
- *infitem.mio\_numdexpcones*
- *infitem.mio\_numdpowcones*
- *infitem.mio\_numintconevar*
- *infitem.mio\_numpepcones*
- *infitem.mio\_numppowcones*
- *infitem.mio\_numqcones*
- *infitem.mio\_numrqcones*
- *infitem.mio\_presolved\_numbinconevar*
- *infitem.mio\_presolved\_numcone*
- *infitem.mio\_presolved\_numconevar*
- *infitem.mio\_presolved\_numcontconevar*
- *infitem.mio\_presolved\_numdexpcones*
- *infitem.mio\_presolved\_numdpowcones*
- *infitem.mio\_presolved\_numintconevar*
- *infitem.mio\_presolved\_numpepcones*
- *infitem.mio\_presolved\_numppowcones*
- *infitem.mio\_presolved\_numqcones*
- *infitem.mio\_presolved\_numrqcones*
- *infitem.purify\_dual\_success*
- *infitem.purify\_primal\_success*
- *linfitem.mio\_anz*

## Removed

- `constant.dataformat.xml`
- `constant.dinfitem.mio_heuristic_time`
- `constant.dinfitem.mio_optimizer_time`
- `constant.iinfitem.mio_construct_num_roundings`
- `constant.iinfitem.mio_initial_solution`
- `constant.iinfitem.mio_near_absgap_satisfied`
- `constant.iinfitem.mio_near_relgap_satisfied`
- `constant.liinfitem.mio_sim_maxiter_setbacks`
- `constant.mionodeseltype.hybrid`
- `constant.mionodeseltype.worst`
- `constant.problemtype.geco`
- `constant.prosta.near_dual_feas`
- `constant.prosta.near_prim_and_dual_feas`
- `constant.prosta.near_prim_feas`
- `constant.sensitivitytype.optimal_partition`
- `constant.solsta.near_dual_feas`
- `constant.solsta.near_dual_infeas_cer`
- `constant.solsta.near_integer_optimal`
- `constant.solsta.near_optimal`
- `constant.solsta.near_prim_and_dual_feas`
- `constant.solsta.near_prim_feas`
- `constant.solsta.near_prim_infeas_cer`

## 18.5 Response Codes

### Added

- *`rescode.err_appending_too_big_cone`*
- *`rescode.err_cbf_duplicate_pow_cones`*
- *`rescode.err_cbf_duplicate_pow_star_cones`*
- *`rescode.err_cbf_invalid_dimension_of_cones`*
- *`rescode.err_cbf_invalid_exp_dimension`*
- *`rescode.err_cbf_invalid_number_of_cones`*
- *`rescode.err_cbf_invalid_power`*
- *`rescode.err_cbf_invalid_power_cone_index`*

- `rescode.err_cbf_invalid_power_star_cone_index`
- `rescode.err_cbf_power_cone_is_too_long`
- `rescode.err_cbf_power_cone_mismatch`
- `rescode.err_cbf_power_star_cone_mismatch`
- `rescode.err_cbf_unhandled_power_cone_type`
- `rescode.err_cbf_unhandled_power_star_cone_type`
- `rescode.err_cone_parameter`
- `rescode.err_format_string`
- `rescode.err_invalid_cj`
- `rescode.err_invalid_file_format_for_cfix`
- `rescode.err_invalid_file_format_for_free_constraints`
- `rescode.err_invalid_file_format_for_nonlinear`
- `rescode.err_invalid_file_format_for_ranged_constraints`
- `rescode.err_num_arguments`
- `rescode.err_ptf_format`
- `rescode.err_server_problem_size`
- `rescode.err_shape_is_too_large`
- `rescode.err_slice_size`
- `rescode.err_too_small_a_truncation_value`
- `rescode.wrn_exp_cones_with_variables_fixed_at_zero`
- `rescode.wrn_pow_cones_with_root_fixed_at_zero`

#### Removed

- `rescode.err_cannot_clone_nl`
- `rescode.err_cannot_handle_nl`
- `rescode.err_invalid_accmode`
- `rescode.err_invalid_file_format_for_general_nl`
- `rescode.err_nonlinear_functions_not_allowed`
- `rescode.err_nr_arguments`
- `rescode.err_open_dl`
- `rescode.err_user_func_ret`
- `rescode.err_user_func_ret_data`
- `rescode.err_user_nlo_eval`
- `rescode.err_user_nlo_eval_hessubi`
- `rescode.err_user_nlo_eval_hessubj`

- `rescode.err_user_nlo_func`
- `rescode.trm_mio_near_abs_gap`
- `rescode.trm_mio_near_rel_gap`
- `rescode.wrn_construct_invalid_sol_itg`
- `rescode.wrn_construct_no_sol_itg`
- `rescode.wrn_construct_solution_infeas`
- `rescode.wrn_no_nonlinear_function_write`

# 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. Vandenbergh, and A. Hassibi. A Tutorial on Geometric Programming. *Optimization and Engineering*, 8(1):67–127, 2007. Available at [http://www.stanford.edu/boyd/gp\\_tutorial.html](http://www.stanford.edu/boyd/gp_tutorial.html).
- [Chvatal83] V. Chvátal. *Linear programming*. W.H. Freeman and Company, 1983.
- [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

DataCallback, 420  
DataCallback.callback, 420  
Env, 215  
Env.syrk, 224  
Env.syevd, 223  
Env.syeig, 223  
Env.sparsetriangularsolvedense, 222  
Env.setupthreads, 222  
Env.set\_Stream, 222  
Env.putlicensewait, 222  
Env.putlicensepath, 222  
Env.putlicensedebug, 221  
Env.putlicensecode, 221  
Env.potrf, 221  
Env.linkfiletostream, 221  
Env.licensecleanup, 221  
Env.getversion, 220  
Env.getcodedesc, 220  
Env.getbuildinfo, 220  
Env.gemv, 219  
Env.gemm, 218  
Env.Env, 215  
Env.echointro, 218  
Env.dot, 218  
Env.Dispose, 215  
Env.computesparseseholesky, 217  
Env.checkoutlicense, 216  
Env.checkinlicense, 216  
Env.checkinall, 216  
Env.axy, 216  
ItgSolutionCallback, 419  
ItgSolutionCallback.callback, 419  
Progress, 419  
Progress.progressCB, 419  
Stream, 420  
Stream.streamCB, 420  
Task, 225  
Task.writetask, 320  
Task.writesolution, 320  
Task.writeparamfile, 320  
Task.writejsonsol, 319  
Task.writedata, 319  
Task.updatesolutioninfo, 319  
Task.toconic, 319  
Task.Task, 225  
Task.strtosk, 318  
Task.strtoconetype, 318  
Task.solvewithbasis, 317  
Task.solutionsummary, 317  
Task.solutiondef, 317  
Task.setdefaults, 317  
Task.set\_Stream, 317  
Task.set\_Progress, 316  
Task.set\_ItgSolutionCallback, 316  
Task.set\_InfoCallback, 316  
Task.sensitivityreport, 316  
Task.resizetask, 316  
Task.removevars, 315  
Task.removecons, 315  
Task.removecones, 315  
Task.removebarvars, 314  
Task.readtask, 314  
Task.readsummary, 314  
Task.readsolution, 314  
Task.readptfstring, 314  
Task.readparamfile, 313  
Task.readopfstring, 313  
Task.readlpstring, 313  
Task.readjsonstring, 313  
Task.readdataformat, 313  
Task.readdata, 312  
Task.putyslice, 312  
Task.puty, 312  
Task.putxxslice, 312  
Task.putxx, 311  
Task.putxcslice, 311  
Task.putxc, 311  
Task.putvartypelist, 310  
Task.putvartype, 310  
Task.putvarsolutionj, 310  
Task.putvarname, 310  
Task.putvarboundsliceconst, 309  
Task.putvarboundslice, 309  
Task.putvarboundlistconst, 308  
Task.putvarboundlist, 308  
Task.putvarbound, 308  
Task.puttaskname, 307  
Task.putsuxslice, 307  
Task.putsux, 307  
Task.putsucslice, 306  
Task.putsuc, 306  
Task.putstrparam, 306  
Task.putsolutionyi, 306  
Task.putsolution, 305  
Task.putsnxslice, 305  
Task.putsnx, 304

Task.putslxslice, 304  
 Task.putslx, 304  
 Task.putslcslice, 304  
 Task.putslc, 303  
 Task.putskxslice, 303  
 Task.putskx, 303  
 Task.putskcslice, 302  
 Task.putskc, 302  
 Task.putqobjij, 302  
 Task.putqobj, 301  
 Task.putqconk, 301  
 Task.putqcon, 300  
 Task.putparam, 300  
 Task.putoptserverhost, 299  
 Task.putobjsense, 299  
 Task.putobjname, 299  
 Task.putnastrparam, 299  
 Task.putnaintparam, 299  
 Task.putnadoupam, 298  
 Task.putmaxnumvar, 298  
 Task.putmaxnumqnz, 298  
 Task.putmaxnumcone, 298  
 Task.putmaxnumcon, 297  
 Task.putmaxnumbarvar, 297  
 Task.putmaxnumanz, 297  
 Task.putintparam, 296  
 Task.putdoupam, 296  
 Task.putcslice, 296  
 Task.putconsolutioni, 295  
 Task.putconname, 295  
 Task.putconename, 295  
 Task.putcone, 295  
 Task.putconboundsliceconst, 294  
 Task.putconboundslice, 294  
 Task.putconboundlistconst, 293  
 Task.putconboundlist, 293  
 Task.putconbound, 293  
 Task.putclist, 292  
 Task.putcj, 292  
 Task.putcfix, 292  
 Task.putbarxj, 291  
 Task.putbarvarname, 291  
 Task.putbarsj, 291  
 Task.putbarcj, 290  
 Task.putbarcblocktriplet, 290  
 Task.putbararowlist, 290  
 Task.putbaraijlist, 289  
 Task.putbaraij, 289  
 Task.putbarablocktriplet, 288  
 Task.putatruncatetol, 288  
 Task.putarowlist, 288  
 Task.putarow, 287  
 Task.putaijlist, 287  
 Task.putaij, 286  
 Task.putacollist, 286  
 Task.putacol, 285  
 Task.primalsensitivity, 284  
 Task.primalrepair, 283  
 Task.optimizersummary, 283  
 Task.optimizermt, 283  
 Task.optimize, 282  
 Task.onesolutionsummary, 282  
 Task.linkfiletostream, 282  
 Task.isstrparname, 282  
 Task.isintparname, 281  
 Task.isdoupam, 281  
 Task.inputdata, 280  
 Task.initbasissolve, 279  
 Task.getyslice, 279  
 Task.gety, 279  
 Task.getxxslice, 279  
 Task.getxx, 278  
 Task.getxcslice, 278  
 Task.getxc, 278  
 Task.getvartypelist, 278  
 Task.getvartype, 277  
 Task.getvarnamelen, 277  
 Task.getvarnameindex, 277  
 Task.getvarname, 276  
 Task.getvarboundslice, 276  
 Task.getvarbound, 276  
 Task.gettasknamelen, 275  
 Task.gettaskname, 275  
 Task.getsymmatinfo, 275  
 Task.getsuxslice, 274  
 Task.getsux, 274  
 Task.getsucslice, 274  
 Task.getsuc, 274  
 Task.getstrparamlen, 273  
 Task.getstrparam, 273  
 Task.getsparsesymmat, 273  
 Task.getsolutionslice, 272  
 Task.getsolutioninfo, 271  
 Task.getsolution, 270  
 Task.getsolsta, 270  
 Task.getsnxslice, 269  
 Task.getsnx, 269  
 Task.getslxslice, 269  
 Task.getslx, 268  
 Task.getslcslice, 268  
 Task.getslc, 268  
 Task.getskxslice, 268  
 Task.getskx, 267  
 Task.getskn, 267  
 Task.getskcslice, 267  
 Task.getskc, 267  
 Task.getreducedcosts, 266  
 Task.getqobjij, 266  
 Task.getqobj, 265  
 Task.getqconk, 265  
 Task.getpvviolvar, 264  
 Task.getpvviolcones, 264  
 Task.getpvviolcon, 264  
 Task.getpvviolbarvar, 263  
 Task.getprosta, 263  
 Task.getprobtype, 263

Task.getprimalsolutionnorms, 262  
 Task.getprimalobj, 262  
 Task.getobjsense, 262  
 Task.getobjnamelen, 262  
 Task.getobjname, 261  
 Task.getnumvar, 261  
 Task.getnumsymmat, 261  
 Task.getnumqobjnz, 261  
 Task.getnumqconknz, 261  
 Task.getnumparam, 260  
 Task.getnumintvar, 260  
 Task.getnumconemem, 260  
 Task.getnumcone, 260  
 Task.getnumcon, 259  
 Task.getnumbarvar, 259  
 Task.getnumbarcnz, 259  
 Task.getnumbarblocktriplets, 259  
 Task.getnumbaranz, 258  
 Task.getnumbarablocktriplets, 258  
 Task.getnumanz64, 258  
 Task.getnumanz, 258  
 Task.getmemusage, 258  
 Task.getmaxnumvar, 257  
 Task.getmaxnumqnz, 257  
 Task.getmaxnumcone, 257  
 Task.getmaxnumcon, 257  
 Task.getmaxnumbarvar, 257  
 Task.getmaxnumanz, 256  
 Task.getlintinf, 256  
 Task.getlenbarvarj, 256  
 Task.getintparam, 255  
 Task.getintinf, 255  
 Task.getinfindex, 255  
 Task.getinfeasiblesubproblem, 254  
 Task.getdviolvar, 254  
 Task.getdviolcones, 253  
 Task.getdviolcon, 253  
 Task.getdviolbarvar, 253  
 Task.getdualsolutionnorms, 252  
 Task.getdualobj, 252  
 Task.getdoupparam, 252  
 Task.getdouinf, 251  
 Task.getdimbarvarj, 251  
 Task.getcslice, 251  
 Task.getconnamelen, 250  
 Task.getconnameindex, 250  
 Task.getconname, 250  
 Task.getconenamelen, 249  
 Task.getconenameindex, 249  
 Task.getconename, 249  
 Task.getconeinfo, 248  
 Task.getcone, 248  
 Task.getconboundslice, 248  
 Task.getconbound, 247  
 Task.getclist, 247  
 Task.getcj, 247  
 Task.getcfix, 247  
 Task.getc, 246  
 Task.getbarxslice, 246  
 Task.getbarxj, 246  
 Task.getbarvarnamelen, 245  
 Task.getbarvarnameindex, 245  
 Task.getbarvarname, 245  
 Task.getbarsslice, 244  
 Task.getbarsj, 244  
 Task.getbarcsparsity, 244  
 Task.getbarcidxj, 243  
 Task.getbarcidxinfo, 243  
 Task.getbarcidx, 243  
 Task.getbarcblocktriplet, 242  
 Task.getbarasparsity, 242  
 Task.getbaraidxinfo, 242  
 Task.getbaraidxj, 241  
 Task.getbaraidx, 241  
 Task.getbarablocktriplet, 240  
 Task.getatruncatetol, 240  
 Task.getarowslicetrip, 239  
 Task.getarowslicenumnz, 239  
 Task.getarowslice, 238  
 Task.getarownumnz, 238  
 Task.getarow, 238  
 Task.getapiecenumnz, 237  
 Task.getaij, 237  
 Task.getacolslicetrip, 236  
 Task.getacolslicenumnz, 236  
 Task.getacolslice, 236  
 Task.getacolnumnz, 235  
 Task.getacol, 235  
 Task.generatevarnames, 235  
 Task.generateconnames, 234  
 Task.generateconenames, 234  
 Task.dualsensitivity, 234  
 Task.Dispose, 225  
 Task.deletesolution, 233  
 Task.commitchanges, 233  
 Task.chgvarbound, 233  
 Task.chgconbound, 232  
 Task.checkmem, 232  
 Task.basiscond, 232  
 Task.asyncstop, 231  
 Task.asyncpoll, 231  
 Task.asyncoptimize, 230  
 Task.asyncgetresult, 230  
 Task.appendvars, 230  
 Task.appendsparsesymmatlist, 229  
 Task.appendsparsesymmat, 228  
 Task.appendcons, 228  
 Task.appendconesseq, 228  
 Task.appendconeseq, 227  
 Task.appendcone, 226  
 Task.appendbarvars, 226  
 Task.analyzesolution, 226  
 Task.analyzeproblem, 226  
 Task.analyzenames, 225

## Enumerations

basindtype, 395  
basindtype.reserved, 395  
basindtype.no\_error, 395  
basindtype.never, 395  
basindtype.if\_feasible, 395  
basindtype.always, 395  
boundkey, 395  
boundkey.up, 395  
boundkey.ra, 396  
boundkey.lo, 395  
boundkey.fx, 395  
boundkey.fr, 396  
branchdir, 413  
branchdir.up, 413  
branchdir.root\_lp, 413  
branchdir.pseudocost, 413  
branchdir.near, 413  
branchdir.guided, 413  
branchdir.free, 413  
branchdir.far, 413  
branchdir.down, 413  
callbackcode, 397  
callbackcode.write\_opf, 401  
callbackcode.update\_primal\_simplex\_bi, 401  
callbackcode.update\_primal\_simplex, 401  
callbackcode.update\_primal\_bi, 401  
callbackcode.update\_presolve, 401  
callbackcode.update\_dual\_simplex\_bi, 401  
callbackcode.update\_dual\_simplex, 401  
callbackcode.update\_dual\_bi, 401  
callbackcode.solving\_remote, 401  
callbackcode.read\_opf\_section, 401  
callbackcode.read\_opf, 401  
callbackcode.primal\_simplex, 401  
callbackcode.new\_int\_mio, 401  
callbackcode.intpnt, 401  
callbackcode.im\_simplex\_bi, 401  
callbackcode.im\_simplex, 401  
callbackcode.im\_root\_cutgen, 401  
callbackcode.im\_read, 401  
callbackcode.im\_qo\_reformulate, 400  
callbackcode.im\_primal\_simplex, 400  
callbackcode.im\_primal\_sensitivity, 400  
callbackcode.im\_primal\_bi, 400  
callbackcode.im\_presolve, 400  
callbackcode.im\_order, 400  
callbackcode.im\_mio\_primal\_simplex, 400  
callbackcode.im\_mio\_intpnt, 400  
callbackcode.im\_mio\_dual\_simplex, 400  
callbackcode.im\_mio, 400  
callbackcode.im\_lu, 400  
callbackcode.im\_license\_wait, 400  
callbackcode.im\_intpnt, 400  
callbackcode.im\_full\_convexity\_check, 400  
callbackcode.im\_dual\_simplex, 400  
callbackcode.im\_dual\_sensitivity, 400  
callbackcode.im\_dual\_bi, 400  
callbackcode.im\_conic, 400  
callbackcode.im\_bi, 400  
callbackcode.end\_write, 400  
callbackcode.end\_to\_conic, 399  
callbackcode.end\_simplex\_bi, 399  
callbackcode.end\_simplex, 399  
callbackcode.end\_root\_cutgen, 399  
callbackcode.end\_read, 399  
callbackcode.end\_qcqp\_reformulate, 399  
callbackcode.end\_primal\_simplex\_bi, 399  
callbackcode.end\_primal\_simplex, 399  
callbackcode.end\_primal\_setup\_bi, 399  
callbackcode.end\_primal\_sensitivity, 399  
callbackcode.end\_primal\_repair, 399  
callbackcode.end\_primal\_bi, 399  
callbackcode.end\_presolve, 399  
callbackcode.end\_optimizer, 399  
callbackcode.end\_mio, 399  
callbackcode.end\_license\_wait, 399  
callbackcode.end\_intpnt, 399  
callbackcode.end\_infeas\_ana, 399  
callbackcode.end\_full\_convexity\_check, 399  
callbackcode.end\_dual\_simplex\_bi, 399  
callbackcode.end\_dual\_simplex, 399  
callbackcode.end\_dual\_setup\_bi, 399  
callbackcode.end\_dual\_sensitivity, 399  
callbackcode.end\_dual\_bi, 398  
callbackcode.end\_conic, 398  
callbackcode.end\_bi, 398  
callbackcode.dual\_simplex, 398  
callbackcode.conic, 398  
callbackcode.begin\_write, 398  
callbackcode.begin\_to\_conic, 398  
callbackcode.begin\_simplex\_bi, 398  
callbackcode.begin\_simplex, 398  
callbackcode.begin\_root\_cutgen, 398  
callbackcode.begin\_read, 398  
callbackcode.begin\_qcqp\_reformulate, 398  
callbackcode.begin\_primal\_simplex\_bi, 398  
callbackcode.begin\_primal\_simplex, 398  
callbackcode.begin\_primal\_setup\_bi, 398  
callbackcode.begin\_primal\_sensitivity, 398  
callbackcode.begin\_primal\_repair, 398  
callbackcode.begin\_primal\_bi, 398  
callbackcode.begin\_presolve, 398  
callbackcode.begin\_optimizer, 398  
callbackcode.begin\_mio, 398  
callbackcode.begin\_license\_wait, 398  
callbackcode.begin\_intpnt, 398  
callbackcode.begin\_infeas\_ana, 398  
callbackcode.begin\_full\_convexity\_check,  
397  
callbackcode.begin\_dual\_simplex\_bi, 397  
callbackcode.begin\_dual\_simplex, 397  
callbackcode.begin\_dual\_setup\_bi, 397  
callbackcode.begin\_dual\_sensitivity, 397  
callbackcode.begin\_dual\_bi, 397  
callbackcode.begin\_conic, 397

callbackcode.begin\_bi, 397  
 checkconvexitytype, 401  
 checkconvexitytype.simple, 401  
 checkconvexitytype.none, 401  
 checkconvexitytype.full, 401  
 compresstype, 402  
 compresstype.zstd, 402  
 compresstype.none, 402  
 compresstype.gzip, 402  
 compresstype.free, 402  
 conetype, 402  
 conetype.zero, 402  
 conetype.rquad, 402  
 conetype.quad, 402  
 conetype.ppow, 402  
 conetype.pexp, 402  
 conetype.dpow, 402  
 conetype.dexp, 402  
 dataformat, 403  
 dataformat.task, 403  
 dataformat.ptf, 403  
 dataformat.op, 403  
 dataformat.mps, 403  
 dataformat.lp, 403  
 dataformat.json\_task, 403  
 dataformat.free\_mps, 403  
 dataformat.extension, 403  
 dataformat.cb, 403  
 dinfitem, 403  
 dinfitem.to\_conic\_time, 407  
 dinfitem.sol\_itr\_pviolvar, 407  
 dinfitem.sol\_itr\_pviolcones, 407  
 dinfitem.sol\_itr\_pviolcon, 407  
 dinfitem.sol\_itr\_pviolbarvar, 407  
 dinfitem.sol\_itr\_primal\_obj, 407  
 dinfitem.sol\_itr\_nrm\_y, 407  
 dinfitem.sol\_itr\_nrm\_xx, 407  
 dinfitem.sol\_itr\_nrm\_xc, 407  
 dinfitem.sol\_itr\_nrm\_sux, 407  
 dinfitem.sol\_itr\_nrm\_suc, 407  
 dinfitem.sol\_itr\_nrm\_snx, 407  
 dinfitem.sol\_itr\_nrm\_slx, 407  
 dinfitem.sol\_itr\_nrm\_slc, 407  
 dinfitem.sol\_itr\_nrm\_barx, 407  
 dinfitem.sol\_itr\_nrmBars, 407  
 dinfitem.sol\_itr\_dviolvar, 407  
 dinfitem.sol\_itr\_dviolcones, 407  
 dinfitem.sol\_itr\_dviolcon, 406  
 dinfitem.sol\_itr\_dviolbarvar, 406  
 dinfitem.sol\_itr\_dual\_obj, 406  
 dinfitem.sol\_itg\_pviolvar, 406  
 dinfitem.sol\_itg\_pviolitg, 406  
 dinfitem.sol\_itg\_pviolcones, 406  
 dinfitem.sol\_itg\_pviolcon, 406  
 dinfitem.sol\_itg\_pviolbarvar, 406  
 dinfitem.sol\_itg\_primal\_obj, 406  
 dinfitem.sol\_itg\_nrm\_xx, 406  
 dinfitem.sol\_itg\_nrm\_xc, 406  
 dinfitem.sol\_itg\_nrm\_barx, 406  
 dinfitem.sol\_bas\_pviolvar, 406  
 dinfitem.sol\_bas\_pviolcon, 406  
 dinfitem.sol\_bas\_primal\_obj, 406  
 dinfitem.sol\_bas\_nrm\_y, 406  
 dinfitem.sol\_bas\_nrm\_xx, 406  
 dinfitem.sol\_bas\_nrm\_xc, 406  
 dinfitem.sol\_bas\_nrm\_sux, 406  
 dinfitem.sol\_bas\_nrm\_suc, 406  
 dinfitem.sol\_bas\_nrm\_slx, 405  
 dinfitem.sol\_bas\_nrm\_slc, 405  
 dinfitem.sol\_bas\_nrm\_barx, 405  
 dinfitem.sol\_bas\_dviolvar, 405  
 dinfitem.sol\_bas\_dviolcon, 405  
 dinfitem.sol\_bas\_dual\_obj, 405  
 dinfitem.sim\_time, 405  
 dinfitem.sim\_primal\_time, 405  
 dinfitem.sim\_obj, 405  
 dinfitem.sim\_feas, 405  
 dinfitem.sim\_dual\_time, 405  
 dinfitem.rd\_time, 405  
 dinfitem.qcqp\_reformulate\_worst\_cholesky\_diag\_scaling, 405  
 dinfitem.qcqp\_reformulate\_worst\_cholesky\_column\_scaling, 405  
 dinfitem.qcqp\_reformulate\_time, 405  
 dinfitem.qcqp\_reformulate\_max\_perturbation, 405  
 dinfitem.primal\_repair\_penalty\_obj, 405  
 dinfitem.presolve\_time, 405  
 dinfitem.presolve\_lindep\_time, 405  
 dinfitem.presolve\_eli\_time, 405  
 dinfitem.optimizer\_time, 405  
 dinfitem.mio\_user\_obj\_cut, 405  
 dinfitem.mio\_time, 405  
 dinfitem.mio\_root\_presolve\_time, 404  
 dinfitem.mio\_root\_optimizer\_time, 404  
 dinfitem.mio\_root\_cutgen\_time, 404  
 dinfitem.mio\_probing\_time, 404  
 dinfitem.mio\_obj\_rel\_gap, 404  
 dinfitem.mio\_obj\_int, 404  
 dinfitem.mio\_obj\_bound, 404  
 dinfitem.mio\_obj\_abs\_gap, 404  
 dinfitem.mio\_knapsack\_cover\_separation\_time, 404  
 dinfitem.mio\_implied\_bound\_time, 404  
 dinfitem.mio\_gmi\_separation\_time, 404  
 dinfitem.mio\_dual\_bound\_after\_presolve, 404  
 dinfitem.mio\_construct\_solution\_obj, 404  
 dinfitem.mio\_cmir\_separation\_time, 404  
 dinfitem.mio\_clique\_separation\_time, 404  
 dinfitem.intpnt\_time, 404  
 dinfitem.intpnt\_primal\_obj, 404  
 dinfitem.intpnt\_primal\_feas, 403  
 dinfitem.intpnt\_order\_time, 403  
 dinfitem.intpnt\_opt\_status, 403  
 dinfitem.intpnt\_factor\_num\_flops, 403  
 dinfitem.intpnt\_dual\_obj, 403

dinfitem.intpnt\_dual\_feas, 403  
 dinfitem.bi\_time, 403  
 dinfitem.bi\_primal\_time, 403  
 dinfitem.bi\_dual\_time, 403  
 dinfitem.bi\_clean\_time, 403  
 dinfitem.bi\_clean\_primal\_time, 403  
 dinfitem.bi\_clean\_dual\_time, 403  
 dparam, 333  
 feature, 407  
 feature.pts, 407  
 feature.pton, 407  
 iinfitem, 408  
 iinfitem.sto\_num\_a\_realloc, 412  
 iinfitem.sol\_itr\_solsta, 412  
 iinfitem.sol\_itr\_prosta, 412  
 iinfitem.sol\_itg\_solsta, 412  
 iinfitem.sol\_itg\_prosta, 412  
 iinfitem.sol\_bas\_solsta, 412  
 iinfitem.sol\_bas\_prosta, 412  
 iinfitem.sim\_solve\_dual, 412  
 iinfitem.sim\_primal\_iter, 412  
 iinfitem.sim\_primal\_inf\_iter, 412  
 iinfitem.sim\_primal\_hotstart\_lu, 412  
 iinfitem.sim\_primal\_hotstart, 412  
 iinfitem.sim\_primal\_deg\_iter, 412  
 iinfitem.sim\_numvar, 412  
 iinfitem.sim\_numcon, 412  
 iinfitem.sim\_dual\_iter, 412  
 iinfitem.sim\_dual\_inf\_iter, 412  
 iinfitem.sim\_dual\_hotstart\_lu, 412  
 iinfitem.sim\_dual\_hotstart, 412  
 iinfitem.sim\_dual\_deg\_iter, 412  
 iinfitem.rd\_prototype, 412  
 iinfitem.rd\_numvar, 411  
 iinfitem.rd\_numq, 411  
 iinfitem.rd\_numintvar, 411  
 iinfitem.rd\_numcone, 411  
 iinfitem.rd\_numcon, 411  
 iinfitem.rd\_numbarvar, 411  
 iinfitem.purify\_primal\_success, 411  
 iinfitem.purify\_dual\_success, 411  
 iinfitem.optimize\_response, 411  
 iinfitem.opt\_numvar, 411  
 iinfitem.opt\_numcon, 411  
 iinfitem.mio\_user\_obj\_cut, 411  
 iinfitem.mio\_total\_num\_cuts, 411  
 iinfitem.mio\_relgap\_satisfied, 411  
 iinfitem.mio\_presolved\_numvar, 411  
 iinfitem.mio\_presolved\_numrqcones, 411  
 iinfitem.mio\_presolved\_numqcones, 411  
 iinfitem.mio\_presolved\_numppowcones, 411  
 iinfitem.mio\_presolved\_numexpcones, 411  
 iinfitem.mio\_presolved\_numintconevar, 411  
 iinfitem.mio\_presolved\_numint, 411  
 iinfitem.mio\_presolved\_numdpowcones, 411  
 iinfitem.mio\_presolved\_numdexpcones, 411  
 iinfitem.mio\_presolved\_numcontconevar, 411  
 iinfitem.mio\_presolved\_numcont, 410  
 iinfitem.mio\_presolved\_numconevar, 410  
 iinfitem.mio\_presolved\_numcone, 410  
 iinfitem.mio\_presolved\_numcon, 410  
 iinfitem.mio\_presolved\_numbinconevar, 410  
 iinfitem.mio\_presolved\_numbin, 410  
 iinfitem.mio\_obj\_bound\_defined, 410  
 iinfitem.mio\_numvar, 410  
 iinfitem.mio\_numrqcones, 410  
 iinfitem.mio\_numqcones, 410  
 iinfitem.mio\_numppowcones, 410  
 iinfitem.mio\_numexpcones, 410  
 iinfitem.mio\_numintconevar, 410  
 iinfitem.mio\_numint, 410  
 iinfitem.mio\_numdpowcones, 410  
 iinfitem.mio\_numdexpcones, 410  
 iinfitem.mio\_numcontconevar, 410  
 iinfitem.mio\_numcont, 410  
 iinfitem.mio\_numconevar, 410  
 iinfitem.mio\_numcone, 410  
 iinfitem.mio\_numcon, 410  
 iinfitem.mio\_numbinconevar, 410  
 iinfitem.mio\_numbin, 410  
 iinfitem.mio\_num\_repeated\_presolve, 410  
 iinfitem.mio\_num\_relax, 409  
 iinfitem.mio\_num\_knapsack\_cover\_cuts, 409  
 iinfitem.mio\_num\_int\_solutions, 409  
 iinfitem.mio\_num\_implied\_bound\_cuts, 409  
 iinfitem.mio\_num\_gomory\_cuts, 409  
 iinfitem.mio\_num\_cmir\_cuts, 409  
 iinfitem.mio\_num\_clique\_cuts, 409  
 iinfitem.mio\_num\_branch, 409  
 iinfitem.mio\_num\_active\_nodes, 409  
 iinfitem.mio\_node\_depth, 409  
 iinfitem.mio\_construct\_solution, 409  
 iinfitem.mio\_clique\_table\_size, 409  
 iinfitem.mio\_absgap\_satisfied, 409  
 iinfitem.intpnt\_solve\_dual, 409  
 iinfitem.intpnt\_num\_threads, 409  
 iinfitem.intpnt\_iter, 409  
 iinfitem.intpnt\_factor\_dim\_dense, 409  
 iinfitem.ana\_pro\_num\_var\_up, 409  
 iinfitem.ana\_pro\_num\_var\_ra, 409  
 iinfitem.ana\_pro\_num\_var\_lo, 409  
 iinfitem.ana\_pro\_num\_var\_int, 409  
 iinfitem.ana\_pro\_num\_var\_fr, 409  
 iinfitem.ana\_pro\_num\_var\_eq, 408  
 iinfitem.ana\_pro\_num\_var\_cont, 408  
 iinfitem.ana\_pro\_num\_var\_bin, 408  
 iinfitem.ana\_pro\_num\_var, 408  
 iinfitem.ana\_pro\_num\_con\_up, 408  
 iinfitem.ana\_pro\_num\_con\_ra, 408  
 iinfitem.ana\_pro\_num\_con\_lo, 408  
 iinfitem.ana\_pro\_num\_con\_fr, 408  
 iinfitem.ana\_pro\_num\_con\_eq, 408  
 iinfitem.ana\_pro\_num\_con, 408  
 inftype, 412  
 inftype.lint\_type, 413  
 inftype.int\_type, 412

- inftype.dou\_type, 412
- intpnthotstart, 397
- intpnthotstart.primal\_dual, 397
- intpnthotstart.primal, 397
- intpnthotstart.none, 397
- intpnthotstart.dual, 397
- iomode, 413
- iomode.write, 413
- iomode.readwrite, 413
- iomode.read, 413
- iparam, 343
- liinfitem, 407
- liinfitem.rd\_numqnz, 408
- liinfitem.rd\_numanz, 408
- liinfitem.mio\_simplex\_iter, 408
- liinfitem.mio\_presolved\_anz, 408
- liinfitem.mio\_intpnt\_iter, 408
- liinfitem.mio\_anz, 408
- liinfitem.intpnt\_factor\_num\_nz, 408
- liinfitem.bi\_primal\_iter, 408
- liinfitem.bi\_dual\_iter, 408
- liinfitem.bi\_clean\_primal\_iter, 408
- liinfitem.bi\_clean\_primal\_deg\_iter, 408
- liinfitem.bi\_clean\_dual\_iter, 408
- liinfitem.bi\_clean\_dual\_deg\_iter, 407
- mark, 396
- mark.up, 396
- mark.lo, 396
- miocontsoltype, 413
- miocontsoltype.root, 413
- miocontsoltype.none, 413
- miocontsoltype.itg\_rel, 413
- miocontsoltype.itg, 413
- miomode, 413
- miomode.satisfied, 413
- miomode.ignored, 413
- mionodeseltype, 414
- mionodeseltype.pseudo, 414
- mionodeseltype.free, 414
- mionodeseltype.first, 414
- mionodeseltype.best, 414
- mpsformat, 414
- mpsformat.strict, 414
- mpsformat.relaxed, 414
- mpsformat.free, 414
- mpsformat.cplex, 414
- nametype, 402
- nametype.mps, 402
- nametype.lp, 402
- nametype.gen, 402
- objsense, 414
- objsense.minimize, 414
- objsense.maximize, 414
- onoffkey, 414
- onoffkey.on, 414
- onoffkey.off, 414
- optimizertype, 414
- optimizertype.primal\_simplex, 414
- optimizertype.mixed\_int, 414
- optimizertype.intpnt, 414
- optimizertype.free\_simplex, 414
- optimizertype.free, 414
- optimizertype.dual\_simplex, 414
- optimizertype.conic, 414
- orderingtype, 414
- orderingtype.try\_graphpar, 415
- orderingtype.none, 415
- orderingtype.free, 414
- orderingtype.force\_graphpar, 415
- orderingtype.experimental, 415
- orderingtype.appminloc, 415
- parametertype, 415
- parametertype.str\_type, 415
- parametertype.invalid\_type, 415
- parametertype.int\_type, 415
- parametertype.dou\_type, 415
- presolvemode, 415
- presolvemode.on, 415
- presolvemode.off, 415
- presolvemode.free, 415
- problemitem, 415
- problemitem.var, 415
- problemitem.cone, 415
- problemitem.con, 415
- problemtyp, 415
- problemtyp.qo, 415
- problemtyp.qcqp, 415
- problemtyp.mixed, 415
- problemtyp.lo, 415
- problemtyp.conic, 415
- prosta, 415
- prosta.unknown, 416
- prosta.prim\_infeas\_or\_unbounded, 416
- prosta.prim\_infeas, 416
- prosta.prim\_feas, 416
- prosta.prim\_and\_dual\_infeas, 416
- prosta.prim\_and\_dual\_feas, 416
- prosta.ill\_posed, 416
- prosta.dual\_infeas, 416
- prosta.dual\_feas, 416
- purify, 397
- purify.primal\_dual, 397
- purify.primal, 397
- purify.none, 397
- purify.dual, 397
- purify.auto, 397
- rescode, 377
- rescodetype, 416
- rescodetype.wrn, 416
- rescodetype.unk, 416
- rescodetype.trm, 416
- rescodetype.ok, 416
- rescodetype.err, 416
- scalingmethod, 416
- scalingmethod.pow2, 416
- scalingmethod.free, 417

- scalingtype, 416
- scalingtype.none, 416
- scalingtype.moderate, 416
- scalingtype.free, 416
- scalingtype.aggressive, 416
- scopr, 402
- scopr.sqrt, 402
- scopr.pow, 402
- scopr.log, 402
- scopr.exp, 402
- scopr.ent, 402
- sensitivitytype, 417
- sensitivitytype.basis, 417
- simdegen, 396
- simdegen.none, 396
- simdegen.moderate, 396
- simdegen.minimum, 396
- simdegen.free, 396
- simdegen.aggressive, 396
- simdupvec, 396
- simdupvec.on, 396
- simdupvec.off, 396
- simdupvec.free, 396
- simhotstart, 397
- simhotstart.status\_keys, 397
- simhotstart.none, 397
- simhotstart.free, 397
- simreform, 396
- simreform.on, 396
- simreform.off, 396
- simreform.free, 396
- simreform.aggressive, 396
- simseltype, 417
- simseltype.se, 417
- simseltype.partial, 417
- simseltype.full, 417
- simseltype.free, 417
- simseltype.devex, 417
- simseltype.ase, 417
- solitem, 417
- solitem.y, 417
- solitem.xx, 417
- solitem.xc, 417
- solitem.sux, 417
- solitem.suc, 417
- solitem.snx, 417
- solitem.slx, 417
- solitem.slc, 417
- solsta, 417
- solsta.unknown, 417
- solsta.prim\_infeas\_cer, 418
- solsta.prim\_illposed\_cer, 418
- solsta.prim\_feas, 417
- solsta.prim\_and\_dual\_feas, 418
- solsta.optimal, 417
- solsta.integer\_optimal, 418
- solsta.dual\_infeas\_cer, 418
- solsta.dual\_illposed\_cer, 418

- solsta.dual\_feas, 417
- soltype, 418
- soltype.itr, 418
- soltype.itg, 418
- soltype.bas, 418
- solveform, 418
- solveform.primal, 418
- solveform.free, 418
- solveform.dual, 418
- sparam, 374
- stakey, 418
- stakey.upr, 418
- stakey.unk, 418
- stakey.supbas, 418
- stakey.low, 418
- stakey.inf, 418
- stakey.fix, 418
- stakey.bas, 418
- startpointtype, 418
- startpointtype.satisfy\_bounds, 419
- startpointtype.guess, 418
- startpointtype.free, 418
- startpointtype.constant, 419
- streamtype, 419
- streamtype.wrn, 419
- streamtype.msg, 419
- streamtype.log, 419
- streamtype.err, 419
- symmatttype, 402
- symmatttype.sparse, 402
- transpose, 396
- transpose.yes, 396
- transpose.no, 396
- uplo, 396
- uplo.up, 396
- uplo.lo, 396
- value, 419
- value.max\_str\_len, 419
- value.license\_buffer\_length, 419
- variabletype, 419
- variabletype.type\_int, 419
- variabletype.type\_cont, 419
- xmlwriteroutputtype, 416
- xmlwriteroutputtype.row, 416
- xmlwriteroutputtype.col, 416

## Exceptions

- ArrayLengthException, 321
- Error, 320
- Exception, 320
- MosekException, 320
- NullArrayException, 321
- Warning, 320

## Parameters

- Double parameters, 333
- dparam.ana\_sol\_infeas\_tol, 333
- dparam.basis\_rel\_tol\_s, 333

dparam.basis\_tol\_s, 333  
dparam.basis\_tol\_x, 333  
dparam.check\_convexity\_rel\_tol, 333  
dparam.data\_sym\_mat\_tol, 333  
dparam.data\_sym\_mat\_tol\_huge, 334  
dparam.data\_sym\_mat\_tol\_large, 334  
dparam.data\_tol\_aj\_huge, 334  
dparam.data\_tol\_aj\_large, 334  
dparam.data\_tol\_bound\_inf, 334  
dparam.data\_tol\_bound\_wrn, 334  
dparam.data\_tol\_c\_huge, 335  
dparam.data\_tol\_cj\_large, 335  
dparam.data\_tol\_qij, 335  
dparam.data\_tol\_x, 335  
dparam.intpnt\_co\_tol\_dfeas, 335  
dparam.intpnt\_co\_tol\_infeas, 335  
dparam.intpnt\_co\_tol\_mu\_red, 336  
dparam.intpnt\_co\_tol\_near\_rel, 336  
dparam.intpnt\_co\_tol\_pfeas, 336  
dparam.intpnt\_co\_tol\_rel\_gap, 336  
dparam.intpnt\_qo\_tol\_dfeas, 336  
dparam.intpnt\_qo\_tol\_infeas, 337  
dparam.intpnt\_qo\_tol\_mu\_red, 337  
dparam.intpnt\_qo\_tol\_near\_rel, 337  
dparam.intpnt\_qo\_tol\_pfeas, 337  
dparam.intpnt\_qo\_tol\_rel\_gap, 337  
dparam.intpnt\_tol\_dfeas, 337  
dparam.intpnt\_tol\_dsafe, 338  
dparam.intpnt\_tol\_infeas, 338  
dparam.intpnt\_tol\_mu\_red, 338  
dparam.intpnt\_tol\_path, 338  
dparam.intpnt\_tol\_pfeas, 338  
dparam.intpnt\_tol\_psafe, 338  
dparam.intpnt\_tol\_rel\_gap, 339  
dparam.intpnt\_tol\_rel\_step, 339  
dparam.intpnt\_tol\_step\_size, 339  
dparam.lower\_obj\_cut, 339  
dparam.lower\_obj\_cut\_finite\_trh, 339  
dparam.mio\_max\_time, 340  
dparam.mio\_rel\_gap\_const, 340  
dparam.mio\_tol\_abs\_gap, 340  
dparam.mio\_tol\_abs\_relax\_int, 340  
dparam.mio\_tol\_feas, 340  
dparam.mio\_tol\_rel\_dual\_bound\_improvement, 340  
dparam.mio\_tol\_rel\_gap, 341  
dparam.optimizer\_max\_time, 341  
dparam.presolve\_tol\_abs\_lindep, 341  
dparam.presolve\_tol\_aj, 341  
dparam.presolve\_tol\_rel\_lindep, 341  
dparam.presolve\_tol\_s, 341  
dparam.presolve\_tol\_x, 341  
dparam.qcqp\_reformulate\_rel\_drop\_tol, 342  
dparam.semidefinite\_tol\_approx, 342  
dparam.sim\_lu\_tol\_rel\_piv, 342  
dparam.simplex\_abs\_tol\_piv, 342  
dparam.upper\_obj\_cut, 342  
dparam.upper\_obj\_cut\_finite\_trh, 342  
Integer parameters, 343  
iparam.ana\_sol\_basis, 343  
iparam.ana\_sol\_print\_violated, 343  
iparam.auto\_sort\_a\_before\_opt, 343  
iparam.auto\_update\_sol\_info, 343  
iparam.basis\_solve\_use\_plus\_one, 343  
iparam.bi\_clean\_optimizer, 344  
iparam.bi\_ignore\_max\_iter, 344  
iparam.bi\_ignore\_num\_error, 344  
iparam.bi\_max\_iterations, 344  
iparam.cache\_license, 344  
iparam.check\_convexity, 345  
iparam.compress\_statfile, 345  
iparam.infeas\_generic\_names, 345  
iparam.infeas\_prefer\_primal, 345  
iparam.infeas\_report\_auto, 345  
iparam.infeas\_report\_level, 345  
iparam.intpnt\_basis, 346  
iparam.intpnt\_diff\_step, 346  
iparam.intpnt\_hotstart, 346  
iparam.intpnt\_max\_iterations, 346  
iparam.intpnt\_max\_num\_cor, 346  
iparam.intpnt\_max\_num\_refinement\_steps, 346  
iparam.intpnt\_multi\_thread, 347  
iparam.intpnt\_off\_col\_trh, 347  
iparam.intpnt\_order\_gp\_num\_seeds, 347  
iparam.intpnt\_order\_method, 347  
iparam.intpnt\_purify, 347  
iparam.intpnt\_regularization\_use, 347  
iparam.intpnt\_scaling, 348  
iparam.intpnt\_solve\_form, 348  
iparam.intpnt\_starting\_point, 348  
iparam.license\_debug, 348  
iparam.license\_pause\_time, 348  
iparam.license\_suppress\_expire\_wrns, 348  
iparam.license\_trh\_expiry\_wrn, 349  
iparam.license\_wait, 349  
iparam.log, 349  
iparam.log\_ana\_pro, 349  
iparam.log\_bi, 349  
iparam.log\_bi\_freq, 350  
iparam.log\_check\_convexity, 350  
iparam.log\_cut\_second\_opt, 350  
iparam.log\_expand, 350  
iparam.log\_feas\_repair, 350  
iparam.log\_file, 350  
iparam.log\_include\_summary, 351  
iparam.log\_infeas\_ana, 351  
iparam.log\_intpnt, 351  
iparam.log\_local\_info, 351  
iparam.log\_mio, 351  
iparam.log\_mio\_freq, 351  
iparam.log\_order, 352  
iparam.log\_presolve, 352  
iparam.log\_response, 352  
iparam.log\_sensitivity, 352  
iparam.log\_sensitivity\_opt, 352  
iparam.log\_sim, 353

- iparam.log\_sim\_freq, 353
- iparam.log\_sim\_minor, 353
- iparam.log\_storage, 353
- iparam.max\_num\_warnings, 353
- iparam.mio\_branch\_dir, 353
- iparam.mio\_conic\_outer\_approximation, 354
- iparam.mio\_cut\_clique, 354
- iparam.mio\_cut\_cmir, 354
- iparam.mio\_cut\_gmi, 354
- iparam.mio\_cut\_implied\_bound, 354
- iparam.mio\_cut\_knapsack\_cover, 354
- iparam.mio\_cut\_selection\_level, 354
- iparam.mio\_feaspump\_level, 355
- iparam.mio\_heuristic\_level, 355
- iparam.mio\_max\_num\_branches, 355
- iparam.mio\_max\_num\_relaxs, 355
- iparam.mio\_max\_num\_root\_cut\_rounds, 356
- iparam.mio\_max\_num\_solutions, 356
- iparam.mio\_mode, 356
- iparam.mio\_node\_optimizer, 356
- iparam.mio\_node\_selection, 356
- iparam.mio\_perspective\_reformulate, 356
- iparam.mio\_probing\_level, 357
- iparam.mio\_propagate\_objective\_constraint, 357
- iparam.mio\_rins\_max\_nodes, 357
- iparam.mio\_root\_optimizer, 357
- iparam.mio\_root\_repeat\_presolve\_level, 357
- iparam.mio\_seed, 358
- iparam.mio\_vb\_detection\_level, 358
- iparam.mt\_spincount, 358
- iparam.num\_threads, 358
- iparam.opf\_write\_header, 358
- iparam.opf\_write\_hints, 359
- iparam.opf\_write\_line\_length, 359
- iparam.opf\_write\_parameters, 359
- iparam.opf\_write\_problem, 359
- iparam.opf\_write\_sol\_bas, 359
- iparam.opf\_write\_sol\_itg, 359
- iparam.opf\_write\_sol\_itr, 360
- iparam.opf\_write\_solutions, 360
- iparam.optimizer, 360
- iparam.param\_read\_case\_name, 360
- iparam.param\_read\_ign\_error, 360
- iparam.presolve\_eliminator\_max\_fill, 360
- iparam.presolve\_eliminator\_max\_num\_tries, 361
- iparam.presolve\_level, 361
- iparam.presolve\_lindep\_abs\_work\_trh, 361
- iparam.presolve\_lindep\_rel\_work\_trh, 361
- iparam.presolve\_lindep\_use, 361
- iparam.presolve\_max\_num\_pass, 361
- iparam.presolve\_max\_num\_reductions, 361
- iparam.presolve\_use, 362
- iparam.primal\_repair\_optimizer, 362
- iparam.ptf\_write\_transform, 362
- iparam.read\_debug, 362
- iparam.read\_keep\_free\_con, 362
- iparam.read\_lp\_drop\_new\_vars\_in\_bou, 363
- iparam.read\_lp\_quoted\_names, 363
- iparam.read\_mps\_format, 363
- iparam.read\_mps\_width, 363
- iparam.read\_task\_ignore\_param, 363
- iparam.remove\_unused\_solutions, 363
- iparam.sensitivity\_all, 364
- iparam.sensitivity\_optimizer, 364
- iparam.sensitivity\_type, 364
- iparam.sim\_basis\_factor\_use, 364
- iparam.sim\_degen, 364
- iparam.sim\_dual\_crash, 364
- iparam.sim\_dual\_phaseone\_method, 365
- iparam.sim\_dual\_restrict\_selection, 365
- iparam.sim\_dual\_selection, 365
- iparam.sim\_exploit\_dupvec, 365
- iparam.sim\_hotstart, 365
- iparam.sim\_hotstart\_lu, 365
- iparam.sim\_max\_iterations, 366
- iparam.sim\_max\_num\_setbacks, 366
- iparam.sim\_non\_singular, 366
- iparam.sim\_primal\_crash, 366
- iparam.sim\_primal\_phaseone\_method, 366
- iparam.sim\_primal\_restrict\_selection, 366
- iparam.sim\_primal\_selection, 367
- iparam.sim\_refactor\_freq, 367
- iparam.sim\_reformulation, 367
- iparam.sim\_save\_lu, 367
- iparam.sim\_scaling, 367
- iparam.sim\_scaling\_method, 367
- iparam.sim\_seed, 368
- iparam.sim\_solve\_form, 368
- iparam.sim\_stability\_priority, 368
- iparam.sim\_switch\_optimizer, 368
- iparam.sol\_filter\_keep\_basic, 368
- iparam.sol\_filter\_keep\_ranged, 368
- iparam.sol\_read\_name\_width, 369
- iparam.sol\_read\_width, 369
- iparam.solution\_callback, 369
- iparam.timing\_level, 369
- iparam.write\_bas\_constraints, 369
- iparam.write\_bas\_head, 369
- iparam.write\_bas\_variables, 370
- iparam.write\_compression, 370
- iparam.write\_data\_param, 370
- iparam.write\_free\_con, 370
- iparam.write\_generic\_names, 370
- iparam.write\_generic\_names\_io, 370
- iparam.write\_ignore\_incompatible\_items, 370
- iparam.write\_int\_constraints, 371
- iparam.write\_int\_head, 371
- iparam.write\_int\_variables, 371
- iparam.write\_lp\_full\_obj, 371
- iparam.write\_lp\_line\_width, 371
- iparam.write\_lp\_quoted\_names, 371
- iparam.write\_lp\_strict\_format, 372
- iparam.write\_lp\_terms\_per\_line, 372
- iparam.write\_mps\_format, 372

- iparam.write\_mps\_int, 372
- iparam.write\_precision, 372
- iparam.write\_sol\_barvariables, 372
- iparam.write\_sol\_constraints, 373
- iparam.write\_sol\_head, 373
- iparam.write\_sol\_ignore\_invalid\_names, 373
- iparam.write\_sol\_variables, 373
- iparam.write\_task\_inc\_sol, 373
- iparam.write\_xml\_mode, 373
- String parameters, 374
- sparam.bas\_sol\_file\_name, 374
- sparam.data\_file\_name, 374
- sparam.debug\_file\_name, 374
- sparam.int\_sol\_file\_name, 374
- sparam.itr\_sol\_file\_name, 374
- sparam.mio\_debug\_string, 374
- sparam.param\_comment\_sign, 374
- sparam.param\_read\_file\_name, 375
- sparam.param\_write\_file\_name, 375
- sparam.read\_mps\_bou\_name, 375
- sparam.read\_mps\_obj\_name, 375
- sparam.read\_mps\_ran\_name, 375
- sparam.read\_mps\_rhs\_name, 375
- sparam.remote\_access\_token, 375
- sparam.sensitivity\_file\_name, 376
- sparam.sensitivity\_res\_file\_name, 376
- sparam.sol\_filter\_xc\_low, 376
- sparam.sol\_filter\_xc\_upr, 376
- sparam.sol\_filter\_xx\_low, 376
- sparam.sol\_filter\_xx\_upr, 376
- sparam.stat\_file\_name, 376
- sparam.stat\_key, 377
- sparam.stat\_name, 377
- sparam.write\_lp\_gen\_var\_name, 377

## Response codes

- Termination, 377
- rescode.ok, 377
- rescode.trm\_internal, 378
- rescode.trm\_internal\_stop, 378
- rescode.trm\_max\_iterations, 377
- rescode.trm\_max\_num\_setbacks, 378
- rescode.trm\_max\_time, 377
- rescode.trm\_mio\_num\_branches, 377
- rescode.trm\_mio\_num\_relaxs, 377
- rescode.trm\_num\_max\_num\_int\_solutions, 377
- rescode.trm\_numerical\_problem, 378
- rescode.trm\_objective\_range, 377
- rescode.trm\_stall, 378
- rescode.trm\_user\_callback, 378
- Warnings, 378
- rescode.wrn\_ana\_almost\_int\_bounds, 380
- rescode.wrn\_ana\_c\_zero, 380
- rescode.wrn\_ana\_close\_bounds, 380
- rescode.wrn\_ana\_empty\_cols, 380
- rescode.wrn\_ana\_large\_bounds, 380
- rescode.wrn\_dropped\_nz\_qobj, 379

- rescode.wrn\_duplicate\_barvariable\_names, 380
- rescode.wrn\_duplicate\_cone\_names, 380
- rescode.wrn\_duplicate\_constraint\_names, 380
- rescode.wrn\_duplicate\_variable\_names, 380
- rescode.wrn\_eliminator\_space, 380
- rescode.wrn\_empty\_name, 379
- rescode.wrn\_exp\_cones\_with\_variables\_fixed\_at\_zero, 380
- rescode.wrn\_ignore\_integer, 379
- rescode.wrn\_incomplete\_linear\_dependency\_check, 379
- rescode.wrn\_large\_aij, 378
- rescode.wrn\_large\_bound, 378
- rescode.wrn\_large\_cj, 378
- rescode.wrn\_large\_con\_fx, 378
- rescode.wrn\_large\_lo\_bound, 378
- rescode.wrn\_large\_up\_bound, 378
- rescode.wrn\_license\_expire, 379
- rescode.wrn\_license\_feature\_expire, 379
- rescode.wrn\_license\_server, 379
- rescode.wrn\_lp\_drop\_variable, 379
- rescode.wrn\_lp\_old\_quad\_format, 378
- rescode.wrn\_mio\_infeasible\_final, 379
- rescode.wrn\_mps\_split\_bou\_vector, 378
- rescode.wrn\_mps\_split\_ran\_vector, 378
- rescode.wrn\_mps\_split\_rhs\_vector, 378
- rescode.wrn\_name\_max\_len, 378
- rescode.wrn\_no\_dualizer, 380
- rescode.wrn\_no\_global\_optimizer, 379
- rescode.wrn\_nz\_in\_upr\_tri, 379
- rescode.wrn\_open\_param\_file, 378
- rescode.wrn\_param\_ignored\_cmio, 379
- rescode.wrn\_param\_name\_dou, 379
- rescode.wrn\_param\_name\_int, 379
- rescode.wrn\_param\_name\_str, 379
- rescode.wrn\_param\_str\_value, 379
- rescode.wrn\_pow\_cones\_with\_root\_fixed\_at\_zero, 380
- rescode.wrn\_presolve\_outofspace, 380
- rescode.wrn\_quad\_cones\_with\_root\_fixed\_at\_zero, 380
- rescode.wrn\_rquad\_cones\_with\_root\_fixed\_at\_zero, 380
- rescode.wrn\_sol\_file\_ignored\_con, 379
- rescode.wrn\_sol\_file\_ignored\_var, 379
- rescode.wrn\_sol\_filter, 379
- rescode.wrn\_spar\_max\_len, 378
- rescode.wrn\_sym\_mat\_large, 380
- rescode.wrn\_too\_few\_basis\_vars, 379
- rescode.wrn\_too\_many\_basis\_vars, 379
- rescode.wrn\_undef\_sol\_file\_name, 379
- rescode.wrn\_using\_generic\_names, 379
- rescode.wrn\_write\_changed\_names, 380
- rescode.wrn\_write\_discarded\_cfix, 380
- rescode.wrn\_zero\_aij, 378
- rescode.wrn\_zeros\_in\_sparse\_col, 379
- rescode.wrn\_zeros\_in\_sparse\_row, 379

Errors, 381  
 rescode.err\_ad\_invalid\_codelist, 392  
 rescode.err\_api\_array\_too\_small, 391  
 rescode.err\_api\_cb\_connect, 391  
 rescode.err\_api\_fatal\_error, 391  
 rescode.err\_api\_internal, 391  
 rescode.err\_appending\_too\_big\_cone, 388  
 rescode.err\_arg\_is\_too\_large, 386  
 rescode.err\_arg\_is\_too\_small, 385  
 rescode.err\_argument\_dimension, 385  
 rescode.err\_argument\_is\_too\_large, 393  
 rescode.err\_argument\_lenneq, 385  
 rescode.err\_argument\_perm\_array, 388  
 rescode.err\_argument\_type, 385  
 rescode.err\_bar\_var\_dim, 392  
 rescode.err\_basis, 387  
 rescode.err\_basis\_factor, 390  
 rescode.err\_basis\_singular, 390  
 rescode.err\_blank\_name, 382  
 rescode.err\_cbf\_duplicate\_acoord, 394  
 rescode.err\_cbf\_duplicate\_bcoord, 394  
 rescode.err\_cbf\_duplicate\_con, 394  
 rescode.err\_cbf\_duplicate\_int, 394  
 rescode.err\_cbf\_duplicate\_obj, 393  
 rescode.err\_cbf\_duplicate\_objacoord, 394  
 rescode.err\_cbf\_duplicate\_pow\_cones, 394  
 rescode.err\_cbf\_duplicate\_pow\_star\_cones, 394  
 rescode.err\_cbf\_duplicate\_psdvar, 394  
 rescode.err\_cbf\_duplicate\_var, 394  
 rescode.err\_cbf\_invalid\_con\_type, 394  
 rescode.err\_cbf\_invalid\_dimension\_of\_cones, 395  
 rescode.err\_cbf\_invalid\_domain\_dimension, 394  
 rescode.err\_cbf\_invalid\_exp\_dimension, 394  
 rescode.err\_cbf\_invalid\_int\_index, 394  
 rescode.err\_cbf\_invalid\_number\_of\_cones, 395  
 rescode.err\_cbf\_invalid\_power, 394  
 rescode.err\_cbf\_invalid\_power\_cone\_index, 394  
 rescode.err\_cbf\_invalid\_power\_star\_cone\_index, 394  
 rescode.err\_cbf\_invalid\_psdvar\_dimension, 394  
 rescode.err\_cbf\_invalid\_var\_type, 394  
 rescode.err\_cbf\_no\_variables, 393  
 rescode.err\_cbf\_no\_version\_specified, 393  
 rescode.err\_cbf\_obj\_sense, 393  
 rescode.err\_cbf\_parse, 393  
 rescode.err\_cbf\_power\_cone\_is\_too\_long, 394  
 rescode.err\_cbf\_power\_cone\_mismatch, 394  
 rescode.err\_cbf\_power\_star\_cone\_mismatch, 394  
 rescode.err\_cbf\_syntax, 393  
 rescode.err\_cbf\_too\_few\_constraints, 394  
 rescode.err\_cbf\_too\_few\_ints, 394  
 rescode.err\_cbf\_too\_few\_psdvar, 394  
 rescode.err\_cbf\_too\_few\_variables, 394  
 rescode.err\_cbf\_too\_many\_constraints, 393  
 rescode.err\_cbf\_too\_many\_ints, 394  
 rescode.err\_cbf\_too\_many\_variables, 393  
 rescode.err\_cbf\_unhandled\_power\_cone\_type, 394  
 rescode.err\_cbf\_unhandled\_power\_star\_cone\_type, 394  
 rescode.err\_cbf\_unsupported, 394  
 rescode.err\_con\_q\_not\_nsd, 388  
 rescode.err\_con\_q\_not\_psd, 387  
 rescode.err\_cone\_index, 388  
 rescode.err\_cone\_overlap, 388  
 rescode.err\_cone\_overlap\_append, 388  
 rescode.err\_cone\_parameter, 388  
 rescode.err\_cone\_rep\_var, 388  
 rescode.err\_cone\_size, 388  
 rescode.err\_cone\_type, 388  
 rescode.err\_cone\_type\_str, 388  
 rescode.err\_data\_file\_ext, 382  
 rescode.err\_dup\_name, 382  
 rescode.err\_duplicate\_aij, 388  
 rescode.err\_duplicate\_barvariable\_names, 393  
 rescode.err\_duplicate\_cone\_names, 393  
 rescode.err\_duplicate\_constraint\_names, 392  
 rescode.err\_duplicate\_variable\_names, 392  
 rescode.err\_end\_of\_file, 382  
 rescode.err\_factor, 390  
 rescode.err\_feasrepair\_cannot\_relax, 390  
 rescode.err\_feasrepair\_inconsistent\_bound, 390  
 rescode.err\_feasrepair\_solving\_relaxed, 390  
 rescode.err\_file\_license, 381  
 rescode.err\_file\_open, 382  
 rescode.err\_file\_read, 382  
 rescode.err\_file\_write, 382  
 rescode.err\_final\_solution, 390  
 rescode.err\_first, 390  
 rescode.err\_firsti, 387  
 rescode.err\_firstj, 387  
 rescode.err\_fixed\_bound\_values, 389  
 rescode.err\_flexlm, 381  
 rescode.err\_format\_string, 382  
 rescode.err\_global\_inv\_conic\_problem, 390  
 rescode.err\_huge\_aij, 388  
 rescode.err\_huge\_c, 388  
 rescode.err\_identical\_tasks, 392  
 rescode.err\_in\_argument, 385  
 rescode.err\_index, 386  
 rescode.err\_index\_arr\_is\_too\_large, 385  
 rescode.err\_index\_arr\_is\_too\_small, 385  
 rescode.err\_index\_is\_too\_large, 385  
 rescode.err\_index\_is\_too\_small, 385  
 rescode.err\_inf\_dou\_index, 385  
 rescode.err\_inf\_dou\_name, 386  
 rescode.err\_inf\_int\_index, 385

rescode.err\_inf\_int\_name, 386  
 rescode.err\_inf\_lint\_index, 385  
 rescode.err\_inf\_lint\_name, 386  
 rescode.err\_inf\_type, 386  
 rescode.err\_infeas\_undefined, 392  
 rescode.err\_infinite\_bound, 388  
 rescode.err\_int64\_to\_int32\_cast, 392  
 rescode.err\_internal, 391  
 rescode.err\_internal\_test\_failed, 392  
 rescode.err\_inv\_aptre, 386  
 rescode.err\_inv\_bk, 386  
 rescode.err\_inv\_bkc, 386  
 rescode.err\_inv\_bkx, 386  
 rescode.err\_inv\_cone\_type, 387  
 rescode.err\_inv\_cone\_type\_str, 387  
 rescode.err\_inv\_marki, 391  
 rescode.err\_inv\_markj, 391  
 rescode.err\_inv\_name\_item, 387  
 rescode.err\_inv\_numi, 391  
 rescode.err\_inv\_numj, 391  
 rescode.err\_inv\_optimizer, 390  
 rescode.err\_inv\_problem, 389  
 rescode.err\_inv\_qcon\_subi, 389  
 rescode.err\_inv\_qcon\_subj, 389  
 rescode.err\_inv\_qcon\_subk, 389  
 rescode.err\_inv\_qcon\_val, 389  
 rescode.err\_inv\_qobj\_subi, 389  
 rescode.err\_inv\_qobj\_subj, 389  
 rescode.err\_inv\_qobj\_val, 389  
 rescode.err\_inv\_sk, 387  
 rescode.err\_inv\_sk\_str, 387  
 rescode.err\_inv\_skc, 387  
 rescode.err\_inv\_skn, 387  
 rescode.err\_inv\_skn, 387  
 rescode.err\_inv\_skn, 387  
 rescode.err\_inv\_var\_type, 386  
 rescode.err\_invalid\_aij, 389  
 rescode.err\_invalid\_ampl\_stub, 392  
 rescode.err\_invalid\_barvar\_name, 383  
 rescode.err\_invalid\_cj, 389  
 rescode.err\_invalid\_compression, 390  
 rescode.err\_invalid\_con\_name, 383  
 rescode.err\_invalid\_cone\_name, 383  
 rescode.err\_invalid\_file\_format\_for\_cfix, 392  
 rescode.err\_invalid\_file\_format\_for\_cones, 392  
 rescode.err\_invalid\_file\_format\_for\_free\_constraints, 392  
 rescode.err\_invalid\_file\_format\_for\_nonlinear\_constraints, 392  
 rescode.err\_invalid\_file\_format\_for\_ranged\_constraints, 392  
 rescode.err\_invalid\_file\_format\_for\_sym\_mat, 392  
 rescode.err\_invalid\_file\_name, 382  
 rescode.err\_invalid\_format\_type, 387  
 rescode.err\_invalid\_idx, 386  
 rescode.err\_invalid\_iomode, 390  
 rescode.err\_invalid\_max\_num, 386  
 rescode.err\_invalid\_name\_in\_sol\_file, 384  
 rescode.err\_invalid\_obj\_name, 382  
 rescode.err\_invalid\_objective\_sense, 389  
 rescode.err\_invalid\_problem\_type, 393  
 rescode.err\_invalid\_sol\_file\_name, 382  
 rescode.err\_invalid\_stream, 382  
 rescode.err\_invalid\_surplus, 387  
 rescode.err\_invalid\_sym\_mat\_dim, 392  
 rescode.err\_invalid\_task, 382  
 rescode.err\_invalid\_utf8, 391  
 rescode.err\_invalid\_var\_name, 383  
 rescode.err\_invalid\_wchar, 391  
 rescode.err\_invalid\_whichsol, 386  
 rescode.err\_json\_data, 385  
 rescode.err\_json\_format, 385  
 rescode.err\_json\_missing\_data, 385  
 rescode.err\_json\_number\_overflow, 385  
 rescode.err\_json\_string, 385  
 rescode.err\_json\_syntax, 385  
 rescode.err\_last, 390  
 rescode.err\_lasti, 387  
 rescode.err\_lastj, 387  
 rescode.err\_lau\_arg\_k, 393  
 rescode.err\_lau\_arg\_m, 393  
 rescode.err\_lau\_arg\_n, 393  
 rescode.err\_lau\_arg\_trans, 393  
 rescode.err\_lau\_arg\_transa, 393  
 rescode.err\_lau\_arg\_transb, 393  
 rescode.err\_lau\_arg\_uplo, 393  
 rescode.err\_lau\_invalid\_lower\_triangular\_matrix, 393  
 rescode.err\_lau\_invalid\_sparse\_symmetric\_matrix, 393  
 rescode.err\_lau\_not\_positive\_definite, 393  
 rescode.err\_lau\_singular\_matrix, 393  
 rescode.err\_lau\_unknown, 393  
 rescode.err\_license, 381  
 rescode.err\_license\_cannot\_allocate, 381  
 rescode.err\_license\_cannot\_connect, 381  
 rescode.err\_license\_expired, 381  
 rescode.err\_license\_feature, 381  
 rescode.err\_license\_invalid\_hostid, 381  
 rescode.err\_license\_max, 381  
 rescode.err\_license\_moseklm\_daemon, 381  
 rescode.err\_license\_no\_server\_line, 381  
 rescode.err\_license\_no\_server\_support, 381  
 rescode.err\_license\_server, 381  
 rescode.err\_license\_server\_version, 381  
 rescode.err\_license\_version, 381  
 rescode.err\_link\_file\_dll, 382  
 rescode.err\_living\_tasks, 382  
 rescode.err\_lower\_bound\_is\_a\_nan, 388  
 rescode.err\_lp\_dup\_slack\_name, 384  
 rescode.err\_lp\_empty, 384  
 rescode.err\_lp\_file\_format, 384  
 rescode.err\_lp\_format, 384  
 rescode.err\_lp\_free\_constraint, 384

rescode.err\_lp\_incompatible, 384  
 rescode.err\_lp\_invalid\_con\_name, 384  
 rescode.err\_lp\_invalid\_var\_name, 384  
 rescode.err\_lp\_write\_conic\_problem, 384  
 rescode.err\_lp\_write\_geco\_problem, 384  
 rescode.err\_lu\_max\_num\_tries, 391  
 rescode.err\_max\_len\_is\_too\_small, 387  
 rescode.err\_maxnumbarvar, 386  
 rescode.err\_maxnumcon, 386  
 rescode.err\_maxnumcone, 388  
 rescode.err\_maxnumqnz, 386  
 rescode.err\_maxnumvar, 386  
 rescode.err\_mio\_internal, 393  
 rescode.err\_mio\_invalid\_node\_optimizer, 395  
 rescode.err\_mio\_invalid\_root\_optimizer, 395  
 rescode.err\_mio\_no\_optimizer, 390  
 rescode.err\_missing\_license\_file, 381  
 rescode.err\_mixed\_conic\_and\_nl, 390  
 rescode.err\_mps\_cone\_overlap, 383  
 rescode.err\_mps\_cone\_repeat, 383  
 rescode.err\_mps\_cone\_type, 383  
 rescode.err\_mps\_duplicate\_q\_element, 384  
 rescode.err\_mps\_file, 383  
 rescode.err\_mps\_inv\_bound\_key, 383  
 rescode.err\_mps\_inv\_con\_key, 383  
 rescode.err\_mps\_inv\_field, 383  
 rescode.err\_mps\_inv\_marker, 383  
 rescode.err\_mps\_inv\_sec\_name, 383  
 rescode.err\_mps\_inv\_sec\_order, 383  
 rescode.err\_mps\_invalid\_obj\_name, 384  
 rescode.err\_mps\_invalid\_objsense, 384  
 rescode.err\_mps\_mul\_con\_name, 383  
 rescode.err\_mps\_mul\_csec, 383  
 rescode.err\_mps\_mul\_qobj, 383  
 rescode.err\_mps\_mul\_qsec, 383  
 rescode.err\_mps\_no\_objective, 383  
 rescode.err\_mps\_non\_symmetric\_q, 383  
 rescode.err\_mps\_null\_con\_name, 383  
 rescode.err\_mps\_null\_var\_name, 383  
 rescode.err\_mps\_splitting\_var, 383  
 rescode.err\_mps\_tab\_in\_field2, 384  
 rescode.err\_mps\_tab\_in\_field3, 384  
 rescode.err\_mps\_tab\_in\_field5, 384  
 rescode.err\_mps\_undef\_con\_name, 383  
 rescode.err\_mps\_undef\_var\_name, 383  
 rescode.err\_mul\_a\_element, 386  
 rescode.err\_name\_is\_null, 390  
 rescode.err\_name\_max\_len, 390  
 rescode.err\_nan\_in\_blc, 389  
 rescode.err\_nan\_in\_blx, 389  
 rescode.err\_nan\_in\_buc, 389  
 rescode.err\_nan\_in\_bux, 389  
 rescode.err\_nan\_in\_c, 389  
 rescode.err\_nan\_in\_double\_data, 389  
 rescode.err\_negative\_append, 390  
 rescode.err\_negative\_surplus, 390  
 rescode.err\_newer\_dll, 382  
 rescode.err\_no\_barx\_for\_solution, 392  
 rescode.err\_no\_basis\_sol, 390  
 rescode.err\_no\_dual\_for\_itg\_sol, 391  
 rescode.err\_no\_dual\_infeas\_cer, 390  
 rescode.err\_no\_init\_env, 382  
 rescode.err\_no\_optimizer\_var\_type, 390  
 rescode.err\_no\_primal\_infeas\_cer, 390  
 rescode.err\_no\_snx\_for\_bas\_sol, 391  
 rescode.err\_no\_solution\_in\_callback, 391  
 rescode.err\_non\_unique\_array, 393  
 rescode.err\_nonconvex, 387  
 rescode.err\_nonlinear\_equality, 387  
 rescode.err\_nonlinear\_ranged, 387  
 rescode.err\_null\_env, 382  
 rescode.err\_null\_pointer, 382  
 rescode.err\_null\_task, 382  
 rescode.err\_num\_arguments, 385  
 rescode.err\_numconlim, 386  
 rescode.err\_numvarlim, 386  
 rescode.err\_obj\_q\_not\_nsd, 388  
 rescode.err\_obj\_q\_not\_psd, 388  
 rescode.err\_objective\_range, 387  
 rescode.err\_older\_dll, 381  
 rescode.err\_opf\_format, 384  
 rescode.err\_opf\_new\_variable, 384  
 rescode.err\_opf\_premature\_eof, 384  
 rescode.err\_optimizer\_license, 381  
 rescode.err\_overflow, 390  
 rescode.err\_param\_index, 385  
 rescode.err\_param\_is\_too\_large, 385  
 rescode.err\_param\_is\_too\_small, 385  
 rescode.err\_param\_name, 385  
 rescode.err\_param\_name\_dou, 385  
 rescode.err\_param\_name\_int, 385  
 rescode.err\_param\_name\_str, 385  
 rescode.err\_param\_type, 385  
 rescode.err\_param\_value\_str, 385  
 rescode.err\_platform\_not\_licensed, 381  
 rescode.err\_postsolve, 390  
 rescode.err\_pro\_item, 387  
 rescode.err\_prob\_license, 381  
 rescode.err\_ptf\_format, 384  
 rescode.err\_qcon\_subi\_too\_large, 389  
 rescode.err\_qcon\_subi\_too\_small, 389  
 rescode.err\_qcon\_upper\_triangle, 389  
 rescode.err\_qobj\_upper\_triangle, 389  
 rescode.err\_read\_format, 383  
 rescode.err\_read\_lp\_missing\_end\_tag, 384  
 rescode.err\_read\_lp\_nonexisting\_name, 384  
 rescode.err\_remove\_cone\_variable, 388  
 rescode.err\_repair\_invalid\_problem, 390  
 rescode.err\_repair\_optimization\_failed, 390  
 rescode.err\_sen\_bound\_invalid\_lo, 391  
 rescode.err\_sen\_bound\_invalid\_up, 391  
 rescode.err\_sen\_format, 391  
 rescode.err\_sen\_index\_invalid, 391  
 rescode.err\_sen\_index\_range, 391  
 rescode.err\_sen\_invalid\_regexp, 391

rescode.err\_sen\_numerical, 391  
 rescode.err\_sen\_solution\_status, 391  
 rescode.err\_sen\_undef\_name, 391  
 rescode.err\_sen\_unhandled\_problem\_type, 391  
 rescode.err\_server\_connect, 395  
 rescode.err\_server\_problem\_size, 395  
 rescode.err\_server\_protocol, 395  
 rescode.err\_server\_status, 395  
 rescode.err\_server\_token, 395  
 rescode.err\_shape\_is\_too\_large, 385  
 rescode.err\_size\_license, 381  
 rescode.err\_size\_license\_con, 381  
 rescode.err\_size\_license\_intvar, 381  
 rescode.err\_size\_license\_numcores, 392  
 rescode.err\_size\_license\_var, 381  
 rescode.err\_slice\_size, 390  
 rescode.err\_sol\_file\_invalid\_number, 388  
 rescode.err\_solitem, 386  
 rescode.err\_solver\_probtype, 387  
 rescode.err\_space, 382  
 rescode.err\_space\_leaking, 383  
 rescode.err\_space\_no\_info, 383  
 rescode.err\_sym\_mat\_duplicate, 392  
 rescode.err\_sym\_mat\_huge, 389  
 rescode.err\_sym\_mat\_invalid, 389  
 rescode.err\_sym\_mat\_invalid\_col\_index, 392  
 rescode.err\_sym\_mat\_invalid\_row\_index, 392  
 rescode.err\_sym\_mat\_invalid\_value, 392  
 rescode.err\_sym\_mat\_not\_lower\_tringular,  
     392  
 rescode.err\_task\_incompatible, 391  
 rescode.err\_task\_invalid, 391  
 rescode.err\_task\_write, 391  
 rescode.err\_thread\_cond\_init, 382  
 rescode.err\_thread\_create, 382  
 rescode.err\_thread\_mutex\_init, 382  
 rescode.err\_thread\_mutex\_lock, 382  
 rescode.err\_thread\_mutex\_unlock, 382  
 rescode.err\_toconic\_constr\_not\_conic, 395  
 rescode.err\_toconic\_constr\_q\_not\_psd, 395  
 rescode.err\_toconic\_constraint\_fx, 395  
 rescode.err\_toconic\_constraint\_ra, 395  
 rescode.err\_toconic\_objective\_not\_psd, 395  
 rescode.err\_too\_small\_a\_truncation\_value,  
     389  
 rescode.err\_too\_small\_max\_num\_nz, 386  
 rescode.err\_too\_small\_maxnumanz, 386  
 rescode.err\_unb\_step\_size, 392  
 rescode.err\_undef\_solution, 387  
 rescode.err\_undefined\_objective\_sense, 389  
 rescode.err\_unhandled\_solution\_status, 393  
 rescode.err\_unknown, 382  
 rescode.err\_upper\_bound\_is\_a\_nan, 388  
 rescode.err\_upper\_triangle, 393  
 rescode.err\_whichitem\_not\_allowed, 386  
 rescode.err\_whichsol, 386  
 rescode.err\_write\_lp\_format, 384  
 rescode.err\_write\_lp\_non\_unique\_name, 384  
 rescode.err\_write\_mps\_invalid\_name, 384  
 rescode.err\_write\_opf\_invalid\_var\_name, 384  
 rescode.err\_writing\_file, 384  
 rescode.err\_xml\_invalid\_problem\_type, 392  
 rescode.err\_y\_is\_undefined, 389

# Index

## Symbols

.NET Core  
    installation, 9

## A

analysis  
    infeasibility, 190  
attaching  
    streams, 17

## B

basic  
    solution, 73  
basis identification, 105, 177  
basis type  
    sensitivity analysis, 196  
BLAS, 114  
bound  
    constraint, 13, 162, 165  
    linear optimization, 13  
    variable, 13, 162, 165

## C

callback, 83  
cardinality constraints, 64, 150  
CBF format, 450  
ce01  
    example, 37  
certificate, 74  
    dual, 164, 168  
    primal, 164, 167  
Cholesky factorization, 116, 138  
column ordered  
    matrix format, 206  
complementarity, 163, 167  
concurrent optimizer, 157  
cone  
    dual, 166  
    dual exponential, 36  
    exponential, 36  
    power, 33  
    quadratic, 28  
    rotated quadratic, 28  
    semidefinite, 40  
conic exponential optimization, 36  
conic optimization, 28, 33, 36, 165  
    interior-point, 181  
    termination criteria, 182  
conic problem

    example, 29, 33, 37  
conic quadratic optimization, 28  
Conic quadratic reformulation, 118  
constraint  
    bound, 13, 162, 165  
    linear optimization, 13  
    matrix, 13, 162, 165  
    quadratic, 170  
correlation matrix, 127  
covariance matrix, *see* correlation matrix  
cq01  
    example, 29  
cut, 185

## D

defining  
    objective, 17  
determinism, 123  
dual  
    certificate, 164, 168  
    cone, 166  
    feasible, 163  
    infeasible, 163, 164, 168  
    problem, 163, 166, 169  
    solution, 75  
    variable, 163, 166  
duality  
    conic, 166  
    linear, 163  
    semidefinite, 169  
dualizer, 173

## E

efficient frontier, 135  
eliminator, 173  
entropy, 60  
    relative, 61  
error  
    optimization, 73  
errors, 77  
example  
    ce01, 37  
    conic problem, 29, 33, 37  
    cq01, 29  
    lo1, 17  
    pow1, 33  
    q01, 20  
    quadratic objective, 20  
exceptions, 77

- exponential, 60
- exponential cone, 36
- F
- factor model, 138
- feasible
  - dual, 163
  - primal, 162, 175, 181
  - problem, 162
- format, 81
  - CBF, 450
  - json, 470
  - LP, 424
  - MPS, 429
  - OPF, 441
  - PTF, 465
  - sol, 478
  - task, 470
- full
  - vector format, 205
- G
- geometric mean, 60
- geometric programming, 53
- GP, 53
- H
- hot-start, 179
- I
- I/O, 81
- infeasibility, 74, 164, 167
  - analysis, 190
  - linear optimization, 164
  - repair, 190
  - semidefinite, 170
- infeasible
  - dual, 163, 164, 168
  - primal, 162, 164, 167, 175, 182
  - problem, 162, 164, 170
- information item, 83, 84
- installation, 7
  - .NET Core, 9
  - IronPython, 9
  - Mono, 9
  - nmake (*command*), 9
  - requirements, 7
  - troubleshooting, 7
- integer
  - optimizer, 185
  - solution, 73
  - variable, 47
- integer feasible
  - solution, 186
- integer optimization, 47, 185
  - cut, 185
  - initial solution, 51
  - objective bound, 185
  - optimality gap, 187
  - parameter, 47
  - relaxation, 185
  - termination criteria, 186
  - tolerance, 186
- integer optimizer
  - logging, 187
- interior-point
  - conic optimization, 181
  - linear optimization, 174
  - logging, 178, 184
  - optimizer, 174, 181
  - solution, 73
  - termination criteria, 176, 182
- IronPython
  - installation, 9
- J
- json format, 470
- L
- LAPACK, 114
- license, 124
- linear
  - objective, 17
- linear constraint matrix, 13
- linear dependency, 173
- linear optimization, 13, 162
  - bound, 13
  - constraint, 13
  - infeasibility, 164
  - interior-point, 174
  - objective, 13
  - simplex, 179
  - termination criteria, 176, 179
  - variable, 13
- linearity interval, 195
- lol
  - example, 17
- log-sum-exp, 61, 154
- logarithm, 60
- logging, 80
  - integer optimizer, 187
  - interior-point, 178, 184
  - optimizer, 178, 180, 184
  - simplex, 180
- logistic regression, 153
- LP format, 424
- M
- machine learning
  - logistic regression, 153
- market impact cost, 139
- Markowitz
  - model, 126
- Markowitz model, 127
  - portfolio optimization, 126
- matrix

- constraint, 13, 162, 165
  - semidefinite, 40
  - symmetric, 40
- matrix format
  - column ordered, 206
  - row ordered, 206
  - triplets, 206
- memory management, 122
- MIP, *see* integer optimization
- mixed-integer, *see* integer
- mixed-integer optimization, *see* integer optimization
- model
  - Markowitz, 126
  - portfolio optimization, 126
- modeling
  - design, 10
- Mono, 9
  - installation, 9
- monomial, 59
- MPS format, 429
  - free, 440

## N

- near-optimal
  - solution, 186
- nmake (*command*)
  - installation, 9
- norm
  - 1-norm, 58
  - 2-norm, 58
  - p-norm, 59
- numerical issues
  - presolve, 173
  - scaling, 173
  - simplex, 179

## O

- objective, 162, 165
  - defining, 17
  - linear, 17
  - linear optimization, 13
- objective bound, 185
- OPF format, 441
- optimal
  - solution, 74
- optimality gap, 187
- optimization
  - conic, 165
  - conic quadratic, 165
  - error, 73
  - linear, 13, 162
  - semidefinite, 169
- optimizer
  - concurrent, 157
  - determinism, 123
  - integer, 185
  - interior-point, 174, 181

- interrupt, 83
- logging, 178, 180, 184
- parallel, 70
- selection, 173, 174
- simplex, 179

## P

- parallel optimization, 70, 157
- parallelization, 123
- parameter, 81
  - integer optimization, 47
  - simplex, 179
- Pareto optimality, 127
- portfolio optimization
  - cardinality constraints, 64, 150
  - efficient frontier, 135
  - factor model, 138
  - market impact cost, 139
  - Markowitz model, 127
  - model, 126
  - Pareto optimality, 127
  - slippage cost, 139
  - transaction cost, 145
- positive semidefinite, 20
- pow1
  - example, 33
- power, 59
- power cone, 33
- power cone optimization, 33
- presolve, 172
  - eliminator, 173
  - linear dependency check, 173
  - numerical issues, 173
- primal
  - certificate, 164, 167
  - feasible, 162, 175, 181
  - infeasible, 162, 164, 167, 175, 182
  - problem, 163, 166, 169
  - solution, 75, 162
- primal-dual
  - problem, 175, 181
  - solution, 163
- problem
  - dual, 163, 166, 169
  - feasible, 162
  - infeasible, 162, 164, 170
  - load, 81
  - primal, 163, 166, 169
  - primal-dual, 175, 181
  - save, 81
  - status, 73
  - unbounded, 164, 168
- PTF format, 465

## Q

- qo1
  - example, 20
- quadratic

- constraint, 170
- quadratic cone, 28
- quadratic objective
  - example, 20
- quadratic optimization, 170
- quality
  - solution, 187

## R

- regression
  - logistic, 153
- relaxation, 185
- repair
  - infeasibility, 190
- response code, 77
- rotated quadratic cone, 28
- row ordered
  - matrix format, 206

## S

- scaling, 173
- semicontinuous variable, 62
- semidefinite
  - cone, 40
  - infeasibility, 170
  - matrix, 40
  - variable, 40
- semidefinite optimization, 40, 169
- sensitivity analysis, 194
  - basis type, 196
- shadow price, 195
- simplex
  - linear optimization, 179
  - logging, 180
  - numerical issues, 179
  - optimizer, 179
  - parameter, 179
  - termination criteria, 179
- single : .NET Core, 10
- slippage cost, 139
- softplus, 61
- sol format, 478
- solution
  - basic, 73
  - dual, 75
  - file format, 478
  - integer, 73
  - integer feasible, 186
  - interior-point, 73
  - near-optimal, 186
  - optimal, 74
  - primal, 75, 162
  - primal-dual, 163
  - quality, 187
  - retrieve, 73
  - status, 16, 74
- solving linear system, 110
- sparse

- vector format, 205
- sparse vector, 205
- status
  - problem, 73
  - solution, 16, 74
- streams
  - attaching, 17
- symmetric
  - matrix, 40

## T

- task format, 470
- termination, 73
- termination criteria, 83
  - conic optimization, 182
  - integer optimization, 186
  - interior-point, 176, 182
  - linear optimization, 176, 179
  - simplex, 179
  - tolerance, 177, 183, 186
- thread, 123
- time limit, 83
- tolerance
  - integer optimization, 186
  - termination criteria, 177, 183, 186
- transaction cost, 145
- triplets
  - matrix format, 206
- troubleshooting
  - installation, 7

## U

- unbounded
  - problem, 164, 168
- user callback, *see* callback

## V

- variable, 162, 165
  - bound, 13, 162, 165
  - dual, 163, 166
  - integer, 47
  - linear optimization, 13
  - semicontinuous, 62
  - semidefinite, 40
- vector format
  - full, 205
  - sparse, 205