

**The MOSEK Python API manual.
Version 6.0 (Revision 148).**



www.mosek.com

Published by MOSEK ApS, Denmark.

Copyright (c) 1998-2012 MOSEK ApS, Denmark. All rights reserved..

Disclaimer: MOSEK ApS (the author of MOSEK) accepts no responsibility for damages resulting from the use of the MOSEK software and makes no warranty, neither expressed nor implied, including, but not limited to, any implied warranty of fitness for a particular purpose. The software is provided as it is, and you, its user, assume all risks when using it.

Contact information

Phone +45 3917 9907
Fax +45 3917 9823

WEB <http://www.mosek.com>

Email	sales@mosek.com	Sales, pricing, and licensing.
	support@mosek.com	Technical support, questions and bug reports.
	info@mosek.com	Everything else.

Mail MOSEK ApS
C/O Symbion Science Park
Fruebjergvej 3, Box 16
2100 Copenhagen Ø
Denmark

Contents

1	Changes and new features in MOSEK	3
1.1	Compilers used to build MOSEK	3
1.2	General changes	3
1.3	Optimizers	4
1.3.1	Interior point optimizer	4
1.3.2	The simplex optimizers	4
1.3.3	Mixed-integer optimizer	4
1.4	API changes	4
1.5	License system	5
1.6	Other changes	5
1.7	Interfaces	5
1.8	Platform changes	5
2	About this manual	7
3	Getting support and help	9
3.1	MOSEK documentation	9
3.2	Additional reading	9
4	Testing installation and running examples	11
4.1	Microsoft Windows platform	11
4.1.1	Running a Python example	12
4.2	Linux platform	12
4.2.1	Running a Python example	13
4.3	Implementation details	13

5 Basic API tutorial	15
5.1 The basics	15
5.1.1 The environment and the task	15
5.1.2 A simple working example	16
5.1.3 Compiling and running examples	18
5.2 Linear optimization	18
5.2.1 Linear optimization example: <code>lo1</code>	19
5.2.2 Row-wise input	25
5.3 Quadratic optimization	28
5.3.1 Example: Quadratic objective	29
5.3.2 Example: Quadratic constraints	33
5.4 Conic optimization	36
5.4.1 Example: <code>cqo1</code>	37
5.5 Integer optimization	40
5.5.1 Example: <code>mil01</code>	40
5.5.2 Specifying an initial solution	43
5.5.3 Example: Specifying an integer solution	44
5.6 Problem modification and reoptimization	47
5.6.1 A production planning problem	47
5.6.2 Changing the A matrix	49
5.6.3 Appending variables	49
5.6.4 Reoptimization	50
5.6.5 Appending constraints	51
5.7 Efficiency considerations	51
5.7.1 API overhead	53
5.8 Conventions employed in the API	53
5.8.1 Naming conventions for arguments	53
5.8.2 Vector formats	55
5.8.3 Matrix formats	56
5.8.4 Array objects	58
5.8.5 Typical problems using the Python API	58
5.9 The license system	59

5.9.1	Waiting for a free license	59
6	Advanced API tutorial	61
6.1	Linear network flow problems	61
6.1.1	A linear network flow problem example	62
6.2	Embedded network flow problems	65
6.2.1	Example: Exploit embedded network flow structure in the simplex optimizer	66
7	Modelling	71
7.1	Linear optimization	71
7.1.1	Duality for linear optimization	72
7.1.2	Primal and dual infeasible case	74
7.2	Quadratic and quadratically constrained optimization	75
7.2.1	A general recommendation	75
7.2.2	Reformulating as a separable quadratic problem	75
7.3	Conic optimization	77
7.3.1	Duality for conic optimization	78
7.3.2	Infeasibility	78
7.3.3	Examples	78
7.3.4	Potential pitfalls in conic optimization	85
7.4	Recommendations	87
7.4.1	Avoid near infeasible models	88
7.5	Examples continued	88
7.5.1	The absolute value	88
7.5.2	The Markowitz portfolio model	88
8	The optimizers for continuous problems	93
8.1	How an optimizer works	93
8.1.1	Presolve	93
8.1.2	Dualizer	95
8.1.3	Scaling	95
8.1.4	Using multiple CPU's	96
8.2	Linear optimization	96
8.2.1	Optimizer selection	96

8.2.2	The interior-point optimizer	96
8.2.3	The simplex based optimizer	101
8.2.4	The interior-point or the simplex optimizer?	102
8.2.5	The primal or the dual simplex variant?	102
8.3	Linear network optimization	103
8.3.1	Network flow problems	103
8.3.2	Embedded network problems	103
8.4	Conic optimization	104
8.4.1	The interior-point optimizer	104
8.5	Nonlinear convex optimization	104
8.5.1	The interior-point optimizer	104
8.6	Solving problems in parallel	105
8.6.1	Thread safety	106
8.6.2	The parallelized interior-point optimizer	106
8.6.3	The concurrent optimizer	106
8.6.4	A more flexible concurrent optimizer	108
8.7	Understanding solution quality	110
8.7.1	The solution summary	111
8.7.2	Retrieving solution quality information with the API	112
9	The optimizer for mixed integer problems	113
9.1	Some notation	113
9.2	An important fact about integer optimization problems	114
9.3	How the integer optimizer works	114
9.3.1	Presolve	115
9.3.2	Heuristic	115
9.3.3	The optimization phase	115
9.4	Termination criterion	115
9.5	How to speed up the solution process	116
9.6	Understanding solution quality	117
9.6.1	Solutionsummary	117
9.6.2	Retrieving solution quality information with the API	118

10 The analyzers	119
10.1 The problem analyzer	119
10.1.1 General characteristics	120
10.1.2 Objective	121
10.1.3 Linear constraints	122
10.1.4 Constraint and variable bounds	122
10.1.5 Quadratic constraints	122
10.1.6 Conic constraints	123
10.2 Analyzing infeasible problems	123
10.2.1 Example: Primal infeasibility	123
10.2.2 Locating the cause of primal infeasibility	125
10.2.3 Locating the cause of dual infeasibility	125
10.2.4 The infeasibility report	126
10.2.5 Theory concerning infeasible problems	130
10.2.6 The certificate of primal infeasibility	130
10.2.7 The certificate of dual infeasibility	131
11 Primal feasibility repair	133
11.1 The main idea	133
11.2 Feasibility repair in MOSEK	135
11.2.1 Usage of negative weights	135
11.2.2 Automatical naming	135
11.2.3 Feasibility repair using the API	136
11.2.4 An example	136
12 Sensitivity analysis	139
12.1 Introduction	139
12.2 Restrictions	139
12.3 References	139
12.4 Sensitivity analysis for linear problems	140
12.4.1 The optimal objective value function	140
12.4.2 The basis type sensitivity analysis	141
12.4.3 The optimal partition type sensitivity analysis	142

12.4.4 Example: Sensitivity analysis	143
12.5 Sensitivity analysis from the MOSEK API	146
12.6 Sensitivity analysis with the command line tool	149
12.6.1 Sensitivity analysis specification file	150
12.6.2 Example: Sensitivity analysis from command line	151
12.6.3 Controlling log output	152
13 Usage guidelines	153
13.1 Verifying the results	153
13.1.1 Verifying primal feasibility	154
13.1.2 Verifying optimality	154
13.2 Turn on logging	154
13.3 Turn on data checking	155
13.4 Debugging an optimization task	155
13.5 Important API limitations	155
13.5.1 Thread safety	155
13.6 Bug reporting	156
14 API reference	157
14.1 API Functionality	157
14.1.1 Analyzing the problem and associated data	157
14.1.2 Reading and writing data files	157
14.1.3 Solutions	158
14.1.4 Memory allocation and deallocation	159
14.1.5 Changing problem specification	160
14.1.6 Delete problem elements (variables,constraints,cones)	161
14.1.7 Add problem elements (variables,constraints,cones)	161
14.1.8 Problem inspection	162
14.1.9 Conic constraints	164
14.1.10 Bounds	164
14.1.11 Output stream functions	164
14.1.12 Objective function	165
14.1.13 Optimizer statistics	166

14.1.14 Parameters (set/get)	166
14.1.15 Naming	167
14.1.16 Preallocating space for problem data	168
14.1.17 Integer variables	169
14.1.18 Quadratic terms	169
14.1.19 Diagnosing infeasibility	170
14.1.20 Optimization	170
14.1.21 Network optimization	170
14.1.22 Sensitivity analysis	171
14.1.23 Testing data validity	171
14.1.24 Solving with the basis	171
14.1.25 Initialization of environment	171
14.1.26 Change <i>A</i>	172
14.2 Class <code>mosek.ArrayLengthException</code>	172
14.3 Class <code>mosek.Env</code>	172
14.3.1 Constructors	173
14.3.2 Methods	173
14.4 Class <code>mosek.Error</code>	177
14.4.1 Constructors	177
14.5 Class <code>mosek.Exception</code>	177
14.5.1 Constructors	178
14.6 Class <code>mosek.Task</code>	178
14.6.1 Constructors	178
14.6.2 Attributes	178
14.6.3 Methods	179
14.7 Class <code>mosek.Warning</code>	249
14.7.1 Constructors	249
15 Parameter reference	251
15.1 Parameter groups	251
15.1.1 Logging parameters.	251
15.1.2 Basis identification parameters.	253
15.1.3 The Interior-point method parameters.	253

15.1.4 Simplex optimizer parameters.	256
15.1.5 Primal simplex optimizer parameters.	258
15.1.6 Dual simplex optimizer parameters.	258
15.1.7 Network simplex optimizer parameters.	258
15.1.8 Nonlinear convex method parameters.	258
15.1.9 The conic interior-point method parameters.	259
15.1.10 The mixed-integer optimization parameters.	260
15.1.11 Presolve parameters.	262
15.1.12 Termination criterion parameters.	263
15.1.13 Progress call-back parameters.	265
15.1.14 Non-convex solver parameters.	265
15.1.15 Feasibility repair parameters.	265
15.1.16 Optimization system parameters.	265
15.1.17 Output information parameters.	266
15.1.18 Extra information about the optimization problem.	268
15.1.19 Overall solver parameters.	268
15.1.20 Behavior of the optimization task.	269
15.1.21 Data input/output parameters.	270
15.1.22 Analysis parameters.	275
15.1.23 Solution input/output parameters.	275
15.1.24 Infeasibility report parameters.	277
15.1.25 License manager parameters.	277
15.1.26 Data check parameters.	277
15.1.27 Debugging parameters.	278
15.2 Double parameters	279
15.3 Integer parameters	301
15.4 String parameter types	377
16 Response codes	387
17 Constants	409
17.1 Constraint or variable access modes	412
17.2 Function opcode	412

17.3 Function operand type	413
17.4 Basis identification	413
17.5 Bound keys	413
17.6 Specifies the branching direction.	414
17.7 Progress call-back codes	414
17.8 Types of convexity checks.	422
17.9 Compression types	422
17.10Cone types	422
17.11CPU type	422
17.12Data format types	423
17.13Double information items	424
17.14Feasibility repair types	428
17.15License feature	428
17.16Integer information items.	428
17.17Information item types	435
17.18Input/output modes	435
17.19Language selection constants	435
17.20Long integer information items.	435
17.21Mark	436
17.22Continuous mixed-integer solution type	437
17.23Integer restrictions	437
17.24Mixed-integer node selection types	437
17.25MPS file format type	438
17.26Message keys	438
17.27Network detection method	438
17.28Objective sense types	438
17.29On/off	439
17.30Optimizer types	439
17.31Ordering strategies	439
17.32Parameter type	440
17.33Presolve method.	440
17.34Problem data items	440

17.35	Problem types	441
17.36	Problem status keys	441
17.37	Interpretation of quadratic terms in MPS files	442
17.38	Response code type	442
17.39	Scaling type	443
17.40	Scaling type	443
17.41	Sensitivity types	443
17.42	Degeneracy strategies	443
17.43	Exploit duplicate columns.	444
17.44	Hot-start type employed by the simplex optimizer	444
17.45	Problem reformulation.	444
17.46	Simplex selection strategy	444
17.47	Solution items	445
17.48	Solution status keys	445
17.49	Solution types	446
17.50	Solve primal or dual form	447
17.51	Status keys	447
17.52	Starting point types	447
17.53	Stream types	448
17.54	Integer values	448
17.55	Variable types	448
17.56	XML writer output mode	448
A	Problem analyzer examples	451
A.1	air04	451
A.2	arki001	452
A.3	Problem with both linear and quadratic constraints	453
A.4	Problem with both linear and conic constraints	455
B	The MPS file format	457
B.1	The MPS file format	457
B.1.1	An example	459
B.1.2	NAME	459

B.1.3	OBJSENSE (optional)	459
B.1.4	OBJNAME (optional)	460
B.1.5	ROWS	460
B.1.6	COLUMNS	460
B.1.7	RHS (optional)	461
B.1.8	RANGES (optional)	462
B.1.9	QSECTION (optional)	462
B.1.10	BOUNDS (optional)	464
B.1.11	CSECTION (optional)	465
B.1.12	ENDATA	467
B.2	Integer variables	467
B.3	General limitations	468
B.4	Interpretation of the MPS format	468
B.5	The free MPS format	468
C	The LP file format	469
C.1	A warning	469
C.2	The LP file format	469
C.2.1	The sections	470
C.2.2	LP format peculiarities	473
C.2.3	The strict LP format	475
C.2.4	Formatting of an LP file	475
D	The OPF format	477
D.1	Intended use	477
D.2	The file format	477
D.2.1	Sections	478
D.2.2	Numbers	482
D.2.3	Names	482
D.3	Parameters section	483
D.4	Writing OPF files from MOSEK	483
D.5	Examples	484
D.5.1	Linear example <code>1o1.opf</code>	484

D.5.2	Quadratic example <code>qo1.opf</code>	485
D.5.3	Conic quadratic example <code>cqo1.opf</code>	486
D.5.4	Mixed integer example <code>miloi1.opf</code>	487
E	The XML (OSiL) format	489
F	The ORD file format	491
F.1	An example	491
G	The solution file format	493
G.1	The basic and interior solution files	493
G.2	The integer solution file	494

License agreement

Before using the MOSEK software, please read the license agreement available in the distribution at
`mosek\6\license.pdf`

Chapter 1

Changes and new features in MOSEK

The section presents improvements and new features added to MOSEK in version 6.0.

1.1 Compilers used to build MOSEK

MOSEK has been build with the compiler shown in Table 1.1.

Platform	C compiler
linux32x86	Intel C 11.0 (gcc 4.3, glibc 2.3.4)
linux64x86	Intel C 11.0 (gcc 4.3, glibc 2.3.4)
osx32x86	Intel C 11.1 (gcc 4.0)
osx64x86	Intel C 11.1 (gcc 4.0)
solaris32x86	Sun Studio 12
solaris64x86	Sun Studio 12
win32x86	Intel C 11.0 (VS 2005)
win64x86	Intel C 11.0 (VS 2005)

Table 1.1: Compiler version used to build MOSEK

1.2 General changes

- A problem analyzer is now available. It generates an simple report with of statistics and information about the optimization problem and relevant warnings about the problem formulation are included.

- A solution analyzer is now available.
- All timing measures are now wall clock times
- MOSEK employs version 1.2.3 of the zlib library.
- MOSEK employs version 11.6.1 of the FLEXnet licensing tools.
- The convexity of quadratic and quadratic constrained optimization is checked explicitly.
- On Windows all DLLs and EXEs are now signed.
- On all platforms the Jar files are signed.
- MOSEK no longer deals with ctrl-c. The user is responsible for terminating MOSEK in the callback.

1.3 Optimizers

1.3.1 Interior point optimizer

- The speed and stability of interior-point optimizer for linear problems has been improved.
- The speed and stability of the interior-point optimizer for conic problems has been improved. In particular, it is much better at dealing with primal or dual infeasible problems.

1.3.2 The simplex optimizers

- Presolve is now much more effective for simplex optimizers hot-starts.

1.3.3 Mixed-integer optimizer

- The stopping criteria for the mixed-integer optimizer have been changed to conform better with industry standards.

1.4 API changes

- The Mosek/Java API is now built for SUN Java 1.5 and later.
- The Mosek/.NET API is now built for MS .NET 2.0 and later.
- The Mosek/Python API is now based on Python CTypes and uses NumPy instead of Numeric. Python 2.5 and later is supported on all platforms where the `ctypes` module is available.

1.5 License system

- The license conditions have been relaxed, so that a license is shared among all tasks using a single environment. This means that running several optimizations in parallel will only consume one license, as long as the associated tasks share a single MOSEK environment. Please note this is NOT useful when using the MATLAB parallel toolbox.
- By default a license remains checked out for the lifetime of the environment. This behavior can be changed using the parameter `MSK_IPAR.CACHE_LICENSE`.
- Flexlm has been upgraded to version 11.6 from version 11.4.

1.6 Other changes

- The documentation has been improved.

1.7 Interfaces

- The AMPL interface has been augmented so it is possible to pass an initial (feasible) integer solution to mixed-integer optimizer.
- The AMPL interface is now capable of reading the constraint and variable names if they are available.

1.8 Platform changes

- MAC OSX on the PowerPC platform is no longer supported.
- Solaris on the SPARC platform is no longer supported.
- MAC OSX is supported on Intel 64 bit X86 i.e. `osx64x86`.
- Add support for MATLAB R2009b.

Chapter 2

About this manual

This manual covers the general functionality of MOSEK and the usage of the MOSEK Python API.

The MOSEK Python Application Programming Interface makes it possible to access the MOSEK optimizer from any Python application. The whole functionality of the native C API is available through a thin, class-based interface using native Python types and exceptions. All methods in the interface are thin wrappers around functions in the native C API, keeping the overhead induced by the API to a minimum.

The API can be used in Python scripts as well as from the interactive Python command-line. The Python interface is particularly well-suited for fast prototyping of models and for debugging and displaying portions of a problem loaded from a file.

The Python interface consists of one single module, `pymosek`, containing classes and constants used with MOSEK.

New users of the MOSEK Python API are encouraged to read:

- Chapter 4 on compiling and running the distributed examples.
- The relevant parts of Chapter 5, i.e. at least the general introduction and the linear optimization section.
- Chapter 13 for a set of guidelines about developing, testing, and debugging applications employing MOSEK.

This should introduce most of the data structures and functionality necessary to implement and solve an optimization problem.

Chapter 7 contains general material about the mathematical formulations of optimization problems compatible with MOSEK, as well as common tips and tricks for reformulating problems so that they can be solved by MOSEK.

Hence, Chapter 7 is useful when trying to find a good formulation of a specific model.

More advanced examples of modelling and model debugging are located in

- Chapter 11 which deals with analysis of infeasible problems,
- Chapter 12 about the sensitivity analysis interface, and

Finally, the Python API reference material is located in

- Chapter 14 which lists all types and functions,
- Chapter 15 which lists all available parameters,
- Chapter 16 which lists all response codes, and
- Chapter 17 which lists all symbolic constants.

Chapter 3

Getting support and help

3.1 MOSEK documentation

For an overview of the available MOSEK documentation please see

`mosek\6\help\index.html`

in the distribution.

3.2 Additional reading

In this manual it is assumed that the reader is familiar with mathematics and in particular mathematical optimization. Some introduction to linear programming is found in books such as “Linear programming” by Chvátal [13] or “Computer Solution of Linear Programs” by Nazareth [18]. For more theoretical aspects see e.g. “Nonlinear programming: Theory and algorithms” by Bazaraa, Shetty, and Sherali [11]. Finally, the book “Model building in mathematical programming” by Williams [23] provides an excellent introduction to modeling issues in optimization.

Another useful resource is “Mathematical Programming Glossary” available at

<http://glossary.computing.society.informs.org>

Chapter 4

Testing installation and running examples

This chapter describes how to verify that the MOSEK Python API has been installed and works, and how to run the distributed examples.

The MOSEK Python interface, PyMosek, requires a MOSEK installation and a full Python 2.5 or 2.6 installation with `ctypes` support. Furthermore, MOSEK can use arrays from the NumPy package from `scipy.org`.

A Python installer can be obtained from the official site:

<http://www.python.org/>

The NumPy package providing arrays and mathematical functionality can be obtained from:

<http://numpy.scipy.org/>

Please note that starting from MOSEK version 6.0 PyMosek uses the NumPy package rather than the Numeric package. If the NumPy package is not present, the PyMosek package includes a minimal array implementation, `mosek.array`, that can be used instead.

4.1 Microsoft Windows platform

To use the PyMosek interface you will need a working installation of MOSEK and Python. See the MOSEK Installation Manual for instructions on installing and testing MOSEK.

MOSEK includes a binary Python that can be used interactively or for running scripts, but if you have a Python installed on your system you may use that instead.

The PyMosek module is located in one of the directories

```
mosek\6\tools\platform\win64x86\python\2  
mosek\6\tools\platform\win32x86\python\2
```

Examples using the PyMosek interface are found in

```
mosek\6\tools\examples\python
```

4.1.1 Running a Python example

To run one of the distributed examples, open a DOS box and type

```
C:  
cd "\Program Files\mosek\6\tools\examples\python"
```

then to execute example `lo1` type

```
python lo1.py
```

4.2 Linux platform

MOSEK can be installed by unpacked it to some directory — the complete procedure is described in the MOSEK Installation Manual. In the following we assume that MOSEK was unpacked to the user's home directory, and that all steps in the installation was completed.

The PyMosek module is located in

```
mosek/6/tools/platform/$PLATFORM/python/2
```

where `$PLATFORM` is `linux32x86` or `linux64x86`, depending on the platform.

The Python examples are found in

```
mosek/6/tools/examples/python
```

Before the PyMosek API can be used, the system environment must be set correctly up:

- The environment variable `LD_LIBRARY_PATH` must contain the path to the mosek library.
- The environment variable `PYTHONPATH` must point to the location of PyMosek module.

You can verify that this is the case as follows: Open a shell and type

```
echo $PATH
```

This prints a `“:”`-separated list of paths which should contain the path to MOSEK. Then type

```
echo $PYTHONPATH
```

This prints a “:”-separated list of paths which should contain the path to the PyMosek module. If one of the above were missing, they must be set up. This can be done either temporarily or permanently:

- The variables can be temporarily set in current shell. In Bash this can be done by typing

```
export LD_LIBRARY_PATH="$HOME/mosek/6/tools/platform/$PLATFORM/bin:$LD_LIBRARY_PATH"
export PYTHONPATH="$HOME/mosek/6/tools/platform/$PLATFORM/python/2:$PYTHONPATH"
```

Note that other shells may require a different syntax.

- The variables can be permanently set by including following lines in `/.bashrc`

```
if [ -z "$LD_LIBRARY_PATH" ]; then
    export LD_LIBRARY_PATH="$HOME/mosek/6/tools/platform/$PLATFORM/bin"
else
    export LD_LIBRARY_PATH="$HOME/mosek/6/tools/platform/$PLATFORM/bin:$LD_LIBRARY_PATH"
fi

if [ -z "$PYTHONPATH" ]; then
    export PYTHONPATH="$HOME/mosek/6/tools/platform/$PLATFORM/python/2"
else
    export PYTHONPATH="$HOME/mosek/6/tools/platform/$PLATFORM/python/2:$PYTHONPATH"
fi
```

Please note that other shells may use a different syntax.

4.2.1 Running a Python example

To run a Python script, open a shell, change directory to where the script is located, and type

```
python myscript.py
```

where `myscript.py` is the name of the Python script.

4.3 Implementation details

The PyMosek module is implemented as a pure Python module using `ctypes` to call native DLLs. The minimal array module `mosek.array` is implemented as a pure binary DLL plus a pure Python `ctypes`-based module.

This means that the PyMosek module is compatible with Python 2.5 and all later 2.X versions. The new Python 3.X series introduces several language incompatibilities, which means that the module is not immediately compatible with Python 3.X; a Python 3.X compatible version is being developed, but is not be included in the distribution — please contact support@mosek.com if you wish to use Python 3 with MOSEK.

Chapter 5

Basic API tutorial

In this chapter the reader will learn how to build a simple application that uses MOSEK.

A number of examples is provided to demonstrate the functionality required for solving linear, quadratic, and conic problems as well as mixed integer problems.

Please note that the section on linear optimization also describes most of the basic functionality that is not specific to linear problems. Hence, it is recommended to read [Section 5.2](#) before reading the rest of this chapter.

5.1 The basics

A typical program using the MOSEK Python interface can be described shortly:

1. Create a handle to the MOSEK functionality.
2. Create an environment (`mosek.Env`) object.
3. Set up some environment specific data and initialize the environment object.
4. Create a task (`mosek.Task`) object.
5. Load a problem into the task object.
6. Optimize the problem.
7. Fetch the result.

5.1.1 The environment and the task

The first MOSEK related step in any program that employs MOSEK is to create an environment (`mosek.Env`) object. The environment contains environment specific data such as information about

the license file, streams for environment messages etc. Before creating any task objects, the environment must be initialized using `Env.initenv`. When this is done one or more task (`mosek.Task`) objects can be created. Each task is associated with a single environment and defines a complete optimization problem as well as task message streams and optimization parameters.

In Python creation of an environment and a task would look something like this:

```
import mosek

# Create an environment
env = mosek.Env()

# You may connect streams and other callbacks to env here.

# Initialize the environment
env.init()
# Create a task
task = env.Task()

...
# Load a problem into the task, optimize etc.
...
# Fetch a solution from the task.
```

Please note that an environment should, if possible, be shared between multiple tasks.

5.1.2 A simple working example

The following simple example shows a working Python program which

- creates an environment and a task,
- reads a problem from a file,
- optimizes the problem, and
- writes the solution to a file.

```
1 #
2 # Copyright: Copyright (c) 1998-2011 MOSEK ApS, Denmark. All rights reserved.
3 #
4 # File:      simple.py
5 #
6 # Purpose: Demonstrates a very simple example using MOSEK by
7 # reading a problem file, solving the problem and
8 # writing the solution to a file.
9 #
10
```

```

11 import mosek
12 import sys
13
14 if len(sys.argv) <= 1:
15     print "Missing argument. The syntax is:"
16     print " simple inputfile [ solutionfile ]"
17 else:
18     # Make mosek environment.
19     env = mosek.Env ()
20
21     # Initialize the environment.
22     env.init ()
23
24     # Create a task object linked with the environment env.
25     # We create it initially with 0 variables and 0 columns,
26     # since we don't know the size of the problem.
27     task = env.Task (0,0)
28
29     # We assume that a problem file was given as the first command
30     # line argument (received in 'args')
31     task.readdata (sys.argv[1])
32
33     # Solve the problem
34     task.optimize()
35
36     # Print a summary of the solution
37     task.solutionsummary(mosek.streamtype.log)
38
39     # If an output file was specified, write a solution
40     if len(sys.argv) > 2:
41         # We define the output format to be OPF, and tell MOSEK to
42         # leave out parameters and problem data from the output file.
43         task.putintparam (mosek.iparam.write_data_format, mosek.dataformat.opf)
44         task.putintparam (mosek.iparam.opf_write_solutions, mosek.onoffkey.on)
45         task.putintparam (mosek.iparam.opf_write_hints, mosek.onoffkey.off)
46         task.putintparam (mosek.iparam.opf_write_parameters, mosek.onoffkey.off)
47         task.putintparam (mosek.iparam.opf_write_problem, mosek.onoffkey.off)
48
49     task.writedata(sys.argv[2])

```

5.1.2.1 Writing a problem to a file

It is frequently beneficial to write a problem to a file that can be stored for later use or inspected visually. The `Task.writedata` function is used write a problem to a file as follows

```

49 task.writedata(sys.argv[2])

```

By default the extension of the filename is the format written. I.e. the filename `somename.opf` implies the file is written in the OPF format.

Similarly, the function `Task.readdata` reads a problem from a file:

```

31 task.readdata (sys.argv[1])

```

5.1.2.2 Inputting and outputting problem data

An optimization problem consists of several components; objective, objective sense, constraints, variable bounds etc. Therefore, the task (`mosek.Task`) provides a number of methods to operate on the task specific data, all of which are listed in Section 14.6.

5.1.2.3 Setting parameters

Apart from the problem data, the task contains a number of parameters defining the behavior of MOSEK. For example the `mosek.iparam.optimizer` parameter defines which optimizer to use. A complete list of all parameters are listed in Chapter 15.

5.1.3 Compiling and running examples

All examples presented in this chapter are distributed with MOSEK and are available in the directory

`mosek/6/tools/examples/`

in the MOSEK installation. Chapter 4 describes how to compile and run the examples.

It is recommended to copy examples to a different directory before modifying and compiling them.

5.2 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

$$\sum_{j=0}^{n-1} c_j x_j + c^f \quad (5.1)$$

subject to the linear constraints

$$l_k^c \leq \sum_{j=0}^{n-1} a_{kj} x_j \leq u_k^c, \quad k = 0, \dots, m-1, \quad (5.2)$$

and the bounds

$$l_j^x \leq x_j \leq u_j^x, \quad j = 0, \dots, n-1, \quad (5.3)$$

where we have used the problem elements

m and n , which are the number of constraints and variables respectively,

x , which is the variable vector of length n ,

c , which is a coefficient vector of size n

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

c^f , which is a constant,

A , which is a $m \times n$ matrix of coefficients is given by

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

l^c and u^c , which specify the lower and upper bounds on constraints respectively, and

l^x and u^x , which specifies the lower and upper bounds on variables respectively.

Please note the unconventional notation using 0 as the first index rather than 1. Hence, x_0 is the first element in variable vector x . This convention has been adapted from Python arrays which are indexed from 0.

5.2.1 Linear optimization example: lo1

The following is an example of a linear optimization problem:

$$\begin{aligned} & \text{maximize} && 3x_0 + 1x_1 + 5x_2 + 1x_3 \\ & \text{subject to} && 3x_0 + 1x_1 + 2x_2 &= 30, \\ & && 2x_0 + 1x_1 + 3x_2 + 1x_3 &\geq 15, \\ & && & 2x_1 + 3x_3 &\leq 25, \end{aligned} \tag{5.4}$$

having the bounds

$$\begin{aligned} 0 &\leq x_0 \leq \infty, \\ 0 &\leq x_1 \leq 10, \\ 0 &\leq x_2 \leq \infty, \\ 0 &\leq x_3 \leq \infty. \end{aligned} \tag{5.5}$$

5.2.1.1 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. For the complete source code see section 5.2.1.2. The code can also be found in:

mosek\6\tools\examples\python\lo1.py

Create an environment. Before setting up the optimization problem, a MOSEK environment must be created and initialized. This is done in the lines:

```
34 # Make a MOSEK environment
35 env = mosek.Env ()
36 # Attach a printer to the environment
37 env.set_Stream (mosek.streamtype.log, streamprinter)
```

We connect a call-back function to the environment log stream. In this case the call-back function simply prints messages to the standard output stream.

Create an optimization task. Next, an empty task object is created:

```
38 # Create a task
39 task = env.Task(0,0)
40 # Attach a printer to the task
41 task.set_Stream (mosek.streamtype.log, streamprinter)
```

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

Load a problem into the task object. First an estimate of the size of the input data is set. This is done to increase the speed of inputting data and is optional.

```
77 task.putmaxnumvar(NUMVAR)
78 task.putmaxnumcon(NUMCON)
79 task.putmaxnumanz(NUMANZ)
```

Before any problem data can be set, variables and constraints must be added to the problem via calls to the function `Task.append`.

```
80 # Append 'NUMCON' empty constraints.
81 # The constraints will initially have no bounds.
82 task.append(mosek.acemode.con, NUMCON)
83
84 #Append 'NUMVAR' variables.
85 # The variables will initially be fixed at zero (x=0).
86 task.append(mosek.acemode.var, NUMVAR)
```

New variables can now be referenced from other functions with indexes in $0, \dots, \text{numvar} - 1$ and new constraints can be referenced with indexes in $0, \dots, \text{numcon} - 1$. More variables / constraints can be appended later as needed, these will be assigned indexes from `numvar` / `numcon` and up.

Next step is to set the problem data. We loop over each variable index $j = 0, \dots, \text{numvar} - 1$ calling functions to set problem data. We first set the objective coefficient $c_j = c[j]$ by calling the function `Task.putcj`.

Bound key	Type of bound	Lower bound	Upper bound
<code>mosek.boundkey.fx</code>	$\cdots = l_j$	Finite	Identical to the lower bound
<code>mosek.boundkey.fr</code>	Free	Minus infinity	Plus infinity
<code>mosek.boundkey.lo</code>	$l_j \leq \cdots$	Finite	Plus infinity
<code>mosek.boundkey.ra</code>	$l_j \leq \cdots \leq u_j$	Finite	Finite
<code>mosek.boundkey.up</code>	$\cdots \leq u_j$	Minus infinity	Finite

Table 5.1: Interpretation of the bound keys.

```

92 # Set the linear term c_j in the objective.
93 task.putcj(j,c[j])

```

The bounds on variables are stored in the arrays

```

49 # Bound keys for variables
50 bkg = [mosek.boundkey.lo,
51        mosek.boundkey.ra,
52        mosek.boundkey.lo,
53        mosek.boundkey.lo]
54 # Bound values for variables
55 blx = [ 0.0,  0.0,  0.0,  0.0]
56 bux = [+inf, 10.0, +inf, +inf]

```

and are set with calls to `Task.putbound`.

```

94 # Set the bounds on variable j
95 # blx[j] <= x_j <= bux[j]
96 task.putbound(mosek.acemode.var,j,bkg[j],blx[j],bux[j])

```

The *Bound key* stored in `bkg` specify the type of the bound according to Table 5.1. For instance `bkg[0]=mosek.boundkey.lo` means that $x_0 \geq l_0^x$. Finally, the numerical values of the bounds on variables are given by

$$l_j^x = \text{blx}[j] \quad (5.6)$$

and

$$u_j^x = \text{bux}[j]. \quad (5.7)$$

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:

```

63 asub = [ array([0, 1]),
64          array([0, 1, 2]),
65          array([0, 1]),
66          array([1, 2])]
67 aval = [ array([3.0, 2.0]),

```

```

68         array([1.0, 1.0, 2.0]),
69         array([2.0, 3.0]),
70         array([1.0, 3.0]) ]

```

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

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

```

97 # Input column j of A
98 task.putavec(mosek.acemode.var, # Input columns of A.
99             j,                  # Variable (column) index.
100             asub[j],            # Row index of non-zeros in column j.
101             aval[j])            # Non-zero Values of column j.

```

Alternatively, the same A matrix can be set one row at a time; please see section 5.2.2 for an example.

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

```

102 for i in range(NUMCON):
103     task.putbound(mosek.acemode.con, 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`.

```

108 task.optimize()

```

Extracting the solution. After optimizing the status of the solution is examined with a call to `Task.getsolutionstatus`. If the solution status is reported as `mosek.solsta.optimal` or `mosek.solsta.near_optimal` the solution is extracted in the lines below:

```

117 # Output a solution
118 xx = zeros(NUMVAR, float)
119 task.getsolutionslice(mosek.soltype.bas,
120                      mosek.solitem.xx,
121                      0, NUMVAR,
122                      xx)

```

The `Task.getsolutionslice` function obtains a “slice” of 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 `mosek.soltype.bas`. The second argument `mosek.solitem.xx` specifies that we want the variable values of the solution. The two following arguments 0 and NUMVAR specifies the range of variable values we want.

The range specified is the first index (here “0”) up to but not including the second index (here ‘NUMVAR’).

Catching exceptions: We cache any exceptions thrown by mosek in the lines:

```

141 except mosek.Exception, e:
142     print "ERROR: %s" % str(e.errno)
143     if e.msg is not None:
144         print "\t%s" % e.msg
145     sys.exit(1)

```

The types of exceptions that MOSEK can throw can be seen in [14.4](#) and [14.7](#).

5.2.1.2 Source code for lo1

```

1  #
2  # Copyright: Copyright (c) 1998-2011 MOSEK ApS, Denmark. All rights reserved.
3  #
4  # File:      lo1.py
5  #
6  # Purpose: Demonstrates how to solve small linear
7  #           optimization problem using the MOSEK Python API.
8  ##
9
10 import sys
11
12 import mosek
13
14
15 # If numpy is installed, use that, otherwise use the
16 # Mosek's array module.
17 try:
18     from numpy import array,zeros,ones
19 except ImportError:
20     from mosek.array import array, zeros, ones
21
22 # Since the actual value of Infinity is ignored, we define it solely
23 # for symbolic purposes:
24 inf = 0.0
25
26 # Define a stream printer to grab output from MOSEK
27 def streamprinter(text):
28     sys.stdout.write(text)
29     sys.stdout.flush()
30
31 # We might write everything directly as a script, but it looks nicer
32 # to create a function.
33 def main ():
34     # Make a MOSEK environment
35     env = mosek.Env ()
36     # Attach a printer to the environment
37     env.set_Stream (mosek.streamtype.log, streamprinter)
38     # Create a task
39     task = env.Task(0,0)
40     # Attach a printer to the task
41     task.set_Stream (mosek.streamtype.log, streamprinter)
42     # Bound keys for constraints
43     bkc = [mosek.boundkey.fx,
44           mosek.boundkey.lo,
45           mosek.boundkey.up]
46     # Bound values for constraints
47     blc = [30.0, 15.0, -inf]
48     buc = [30.0, +inf, 25.0]
49     # Bound keys for variables
50     bkc = [mosek.boundkey.lo,
51           mosek.boundkey.ra,
52           mosek.boundkey.lo,
53           mosek.boundkey.lo]

```

```

54 # Bound values for variables
55 blx = [ 0.0, 0.0, 0.0, 0.0]
56 bux = [+inf, 10.0, +inf, +inf]
57 # Objective coefficients
58
59 c = [ 3.0, 1.0, 5.0, 1.0 ]
60
61 # Below is the sparse representation of the A
62 # matrix stored by column.
63 asub = [ array([0, 1]),
64         array([0, 1, 2]),
65         array([0, 1]),
66         array([1, 2])]
67 aval = [ array([3.0, 2.0]),
68         array([1.0, 1.0, 2.0]),
69         array([2.0, 3.0]),
70         array([1.0, 3.0]) ]
71 NUMVAR = len(bkx)
72 NUMCON = len(bkc)
73 NUMANZ = 9
74 # Give MOSEK an estimate of the size of the input data.
75 # This is done to increase the speed of inputting data.
76 # However, it is optional.
77 task.putmaxnumvar(NUMVAR)
78 task.putmaxnumcon(NUMCON)
79 task.putmaxnumanz(NUMANZ)
80 # Append 'NUMCON' empty constraints.
81 # The constraints will initially have no bounds.
82 task.append(mosek.acemode.con, NUMCON)
83
84 #Append 'NUMVAR' variables.
85 # The variables will initially be fixed at zero (x=0).
86 task.append(mosek.acemode.var, NUMVAR)
87
88 #Optionally add a constant term to the objective.
89 task.putcfix(0.0)
90
91 for j in range(NUMVAR):
92     # Set the linear term c_j in the objective.
93     task.putcj(j, c[j])
94     # Set the bounds on variable j
95     # blx[j] <= x_j <= bux[j]
96     task.putbound(mosek.acemode.var, j, bkx[j], blx[j], bux[j])
97     # Input column j of A
98     task.putavec(mosek.acemode.var, # Input columns of A.
99                 j, # Variable (column) index.
100                 asub[j], # Row index of non-zeros in column j.
101                 aval[j]) # Non-zero Values of column j.
102 for i in range(NUMCON):
103     task.putbound(mosek.acemode.con, i, bkc[i], blc[i], buc[i])
104 # Input the objective sense (minimize/maximize)
105 task.putobjsense(mosek.objsense.maximize)
106
107 # Optimize the task
108 task.optimize()
109
110 # Print a summary containing information
111 # about the solution for debugging purposes

```

```

112 task.solutionsummary(mosek.streamtype.msg)
113
114 prosta = []
115 solsta = []
116 [prosta,solsta] = task.getsolutionstatus(mosek.soltype.bas)
117 # Output a solution
118 xx = zeros(NUMVAR, float)
119 task.getsolutionslice(mosek.soltype.bas,
120                       mosek.solitem.xx,
121                       0,NUMVAR,
122                       xx)
123 if solsta == mosek.solsta.optimal or solsta == mosek.solsta.near_optimal:
124     print("Optimal solution: %s" % xx)
125 elif solsta == mosek.solsta.dual_infeas_cer:
126     print("Primal or dual infeasibility.\n")
127 elif solsta == mosek.solsta.prim_infeas_cer:
128     print("Primal or dual infeasibility.\n")
129 elif solsta == mosek.solsta.near_dual_infeas_cer:
130     print("Primal or dual infeasibility.\n")
131 elif solsta == mosek.solsta.near_prim_infeas_cer:
132     print("Primal or dual infeasibility.\n")
133 elif mosek.solsta.unknown:
134     print("Unknown solution status")
135 else:
136     print("Other solution status")
137
138 # call the main function
139 try:
140     main ()
141 except mosek.Exception, e:
142     print "ERROR: %s" % str(e.errno)
143     if e.msg is not None:
144         print "\t%s" % e.msg
145         sys.exit(1)
146 except:
147     import traceback
148     traceback.print_exc()
149     sys.exit(1)
150 sys.exit(0)

```

5.2.2 Row-wise input

In the previous example the A matrix is set one column at a time. Alternatively the same matrix can be set one row at a time or the two methods can be mixed as in the example in section 5.6. The following example show how to set the A matrix by rows.

The source code for this example can be found in:

mosek\6\tools\examples\python\lo2.py

```

1 #
2 # Copyright: Copyright (c) 1998-2011 MOSEK ApS, Denmark. All rights reserved.
3 #
4 # File:      lo2.py

```

```

5 #
6 # Purpose: Demonstrates how to solve small linear
7 #           optimization problem using the MOSEK Python API.
8 ##
9
10 import sys
11
12 import mosek
13 # If numpy is installed, use that, otherwise use the
14 # Mosek's array module.
15 try:
16     from numpy import array,zeros,ones
17 except ImportError:
18     from mosek.array import array, zeros, ones
19
20 # Since the actual value of Infinity is ignored, we define it solely
21 # for symbolic purposes:
22 inf = 0.0
23
24 # Define a stream printer to grab output from MOSEK
25 def streamprinter(text):
26     sys.stdout.write(text)
27     sys.stdout.flush()
28
29 # We might write everything directly as a script, but it looks nicer
30 # to create a function.
31 def main():
32     # Make a MOSEK environment
33     env = mosek.Env()
34     # Attach a printer to the environment
35     env.set_Stream(mosek.streamtype.log, streamprinter)
36
37     # Create a task
38     task = env.Task(0,0)
39     # Attach a printer to the task
40     task.set_Stream(mosek.streamtype.log, streamprinter)
41
42     # Bound keys for constraints
43     bkc = [mosek.boundkey.fx,
44           mosek.boundkey.lo,
45           mosek.boundkey.up]
46     # Bound values for constraints
47     blc = [30.0, 15.0, -inf]
48     buc = [30.0, +inf, 25.0]
49     # Bound keys for variables
50     bkx = [mosek.boundkey.lo,
51           mosek.boundkey.ra,
52           mosek.boundkey.lo,
53           mosek.boundkey.lo]
54     # Bound values for variables
55     blx = [0.0, 0.0, 0.0, 0.0]
56     bux = [+inf, 10.0, +inf, +inf]
57     # Objective coefficients
58
59     c = [3.0, 1.0, 5.0, 1.0]
60
61     # We input the A matrix column-wise
62     # asub contains row indexes

```

```

63  asub = [ array([0, 1, 2]),
64           array([0, 1, 2, 3]),
65           array([0, 3])]
66  # acof contains coefficients
67  aval = [ array([3.0, 1.0, 2.0]),
68           array([2.0, 1.0, 3.0, 1.0]),
69           array([2.0, 3.0])]
70  NUMVAR = len(bkx)
71  NUMCON = len(bkc)
72  NUMANZ = 9
73  # Give MOSEK an estimate of the size of the input data.
74  # This is done to increase the speed of inputting data.
75  # However, it is optional.
76  task.putmaxnumvar(NUMVAR)
77  task.putmaxnumcon(NUMCON)
78  task.putmaxnumanz(NUMANZ)
79  # Append 'NUMCON' empty constraints.
80  # The constraints will initially have no bounds.
81  task.append(mosek.acemode.con, NUMCON)
82
83  #Append 'NUMVAR' variables.
84  # The variables will initially be fixed at zero (x=0).
85  task.append(mosek.acemode.var, NUMVAR)
86
87  #Optionally add a constant term to the objective.
88  task.putcfix(0.0)
89
90  for j in range(NUMVAR):
91      # Set the linear term c_j in the objective.
92      task.putcj(j, c[j])
93      # Set the bounds on variable j
94      # blx[j] <= x_j <= bux[j]
95      task.putbound(mosek.acemode.var, j, bkx[j], blx[j], bux[j])
96
97  for i in range(NUMCON):
98      task.putbound(mosek.acemode.con, i, bkc[i], blc[i], buc[i])
99      # Input row i of A
100     task.putavec(mosek.acemode.con,      #Input row of A.
101                  i,                      # Row index.
102                  asub[i],                # Column indexes of non-zeros in row i.
103                  aval[i]);               # Non-zero Values of row i.
104
105
106  # Input the objective sense (minimize/maximize)
107  task.putobjsense(mosek.objsense.maximize)
108
109  # Optimize the task
110  task.optimize()
111
112  # Print a summary containing information
113  # about the solution for debugging purposes
114  task.solutionsummary(mosek.streamtype.msg)
115
116  prosta = []
117  solsta = []
118  [prosta, solsta] = task.getsolutionstatus(mosek.soltype.bas)
119
120  # Output a solution

```

```

121 xx = zeros(NUMVAR, float)
122 task.getsolutionslice(mosek.soltype.bas,
123                       mosek.solitem.xx,
124                       0, NUMVAR,
125                       xx)
126
127 if solsta == mosek.solsta.optimal or solsta == mosek.solsta.near_optimal:
128     print("Optimal solution: %s" % xx)
129 elif solsta == mosek.solsta.dual_infeas_cer:
130     print("Primal or dual infeasibility.\n")
131 elif solsta == mosek.solsta.prim_infeas_cer:
132     print("Primal or dual infeasibility.\n")
133 elif solsta == mosek.solsta.near_dual_infeas_cer:
134     print("Primal or dual infeasibility.\n")
135 elif solsta == mosek.solsta.near_prim_infeas_cer:
136     print("Primal or dual infeasibility.\n")
137 elif mosek.solsta.unknown:
138     print("Unknown solution status")
139 else:
140     print("Other solution status")
141
142 # call the main function
143 try:
144     main ()
145 except mosek.Exception, (code,msg):
146     print "ERROR: %s" % str(code)
147     if msg is not None:
148         print "\t%s" % msg
149         sys.exit(1)
150 except:
151     import traceback
152     traceback.print_exc()
153     sys.exit(1)
154 sys.exit(0)

```

5.3 Quadratic optimization

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

$$\begin{aligned}
 & \text{minimize} && \frac{1}{2}x^T Q^o x + c^T x + c^f \\
 & \text{subject to} && l_k^c \leq \frac{1}{2}x^T Q^k x + \sum_{j=0}^{n-1} a_{k,j} x_j \leq u_k^c, \quad k = 0, \dots, m-1, \\
 & && l^x \leq x \leq u^x, \quad j = 0, \dots, n-1.
 \end{aligned} \tag{5.8}$$

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

$$x^T Q x = 0.5x^T (Q + Q^T)x.$$

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

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

and the k th constraint must be of the form

$$l_k^c \leq \frac{1}{2}x^T Q^k x + \sum_{j=0}^{n-1} a_{k,j} x_j \quad (5.9)$$

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

$$\frac{1}{2}x^T Q^k x + \sum_{j=0}^{n-1} a_{k,j} x_j \leq u_k^c. \quad (5.10)$$

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

5.3.1 Example: Quadratic objective

The following is an example if a quadratic, linearly constrained problem:

$$\begin{aligned} & \text{minimize} && x_1^2 + 0.1x_2^2 + x_3^2 - x_1x_3 - x_2 \\ & \text{subject to} && 1 \leq x_1 + x_2 + x_3 \\ & && x \geq 0 \end{aligned} \quad (5.11)$$

This can be written equivalently as

$$\begin{aligned} & \text{minimize} && 1/2x^T Q^o x + c^T x \\ & \text{subject to} && Ax \geq b \\ & && x \geq 0, \end{aligned} \quad (5.12)$$

where

$$Q^o = \begin{bmatrix} 2 & 0 & -1 \\ 0 & 0.2 & 0 \\ -1 & 0 & 2 \end{bmatrix}, \quad c = \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix}, \quad A = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}, \quad \text{and } b = 1. \quad (5.13)$$

Please note that MOSEK always assumes that there is a $1/2$ in front of the $x^T Q x$ term in the objective. Therefore, the 1 in front of x_0^2 becomes 2 in Q , i.e. $Q_{0,0}^o = 2$.

5.3.1.1 Source code

```

1  ##
2  #   Copyright: Copyright (c) 1998-2011 MOSEK ApS, Denmark. All rights reserved.
3  #
4  #   File:      qo1.py
5  #
6  #   Purpose:   Demonstrate how to solve a quadratic
7  #             optimization problem using the MOSEK Python API.
8  ##
9
10 import sys
11 import os
12
```

```

13 import mosek
14
15 # If numpy is installed, use that, otherwise use the
16 # Mosek's array module.
17 try:
18     from numpy import array,zeros,ones
19 except ImportError:
20     from mosek.array import array, zeros, ones
21
22 # Since the actual value of Infinity is ignored, we define it solely
23 # for symbolic purposes:
24 inf = 0.0
25
26 # Define a stream printer to grab output from MOSEK
27 def streamprinter(text):
28     sys.stdout.write(text)
29     sys.stdout.flush()
30
31 # We might write everything directly as a script, but it looks nicer
32 # to create a function.
33 def main ():
34     # Open MOSEK and create an environment and task
35     # Make a MOSEK environment
36     env = mosek.Env ()
37     # Attach a printer to the environment
38     env.set_Stream (mosek.streamtype.log, streamprinter)
39     # Create a task
40     task = env.Task()
41     task.set_Stream (mosek.streamtype.log, streamprinter)
42     # Set up and input bounds and linear coefficients
43     bkc = [ mosek.boundkey.lo ]
44     blc = [ 1.0 ]
45     buc = [ inf ]
46
47     bkc = [ mosek.boundkey.lo,
48             mosek.boundkey.lo,
49             mosek.boundkey.lo ]
50     blx = [ 0.0, 0.0, 0.0 ]
51     bux = [ inf, inf, inf ]
52     c = [ 0.0, -1.0, 0.0 ]
53     asub = [ array([0]), array([0]), array([0]) ]
54     aval = [ array([1.0]), array([1.0]), array([1.0])]
55
56     NUMVAR = len(bkc)
57     NUMCON = len(bkc)
58     NUMANZ = 3
59
60     # Give MOSEK an estimate of the size of the input data.
61     # This is done to increase the speed of inputting data.
62     # However, it is optional.
63     task.putmaxnumvar(NUMVAR)
64     task.putmaxnumcon(NUMCON)
65     task.putmaxnumanz(NUMANZ)
66     # Append 'NUMCON' empty constraints.
67     # The constraints will initially have no bounds.
68     task.append(mosek.acemode.con, NUMCON)
69
70     #Append 'NUMVAR' variables.

```

```

71 # The variables will initially be fixed at zero (x=0).
72 task.append(mosek.accmode.var, NUMVAR)
73 # Optionally add a constant term to the objective.
74 task.putcfix(0.0)
75
76 for j in range(NUMVAR):
77 # Set the linear term c_j in the objective.
78     task.putcj(j, c[j])
79     # Set the bounds on variable j
80     # blx[j] <= x_j <= bux[j]
81     task.putbound(mosek.accmode.var, j, bkg[j], blx[j], bux[j])
82     # Input column j of A
83     task.putavec(mosek.accmode.var, # Input columns of A.
84                  j, # Variable (column) index.
85                  asub[j], # Row index of non-zeros in column j.
86                  aval[j]) # Non-zero Values of column j.
87 for i in range(NUMCON):
88     task.putbound(mosek.accmode.con, i, bkg[i], blc[i], buc[i])
89
90 # Input the objective sense (minimize/maximize)
91 task.putobjsense(mosek.objsense.maximize)
92
93 # Set up and input quadratic objective
94 qsubi = [ 0, 1, 2, 2 ]
95 qsubj = [ 0, 1, 0, 2 ]
96 qval = [ 2.0, 0.2, -1.0, 2.0 ]
97
98 task.putqobj(qsubi, qsubj, qval)
99
100 task.putobjsense(mosek.objsense.minimize)
101
102 # Optimize
103 task.optimize()
104 # Print a summary containing information
105 # about the solution for debugging purposes
106 task.solutionsummary(mosek.streamtype.msg)
107
108 prosta = []
109 solsta = []
110 [prosta, solsta] = task.getsolutionstatus(mosek.soltype.itr)
111
112 # Output a solution
113 xx = zeros(NUMVAR, float)
114 task.getsolutionslice(mosek.soltype.itr,
115                       mosek.solitem.xx,
116                       0, NUMVAR,
117                       xx)
118
119 if solsta == mosek.solsta.optimal or solsta == mosek.solsta.near_optimal:
120     print("Optimal solution: %s" % xx)
121 elif solsta == mosek.solsta.dual_infeas_cer:
122     print("Primal or dual infeasibility.\n")
123 elif solsta == mosek.solsta.prim_infeas_cer:
124     print("Primal or dual infeasibility.\n")
125 elif solsta == mosek.solsta.near_dual_infeas_cer:
126     print("Primal or dual infeasibility.\n")
127 elif solsta == mosek.solsta.near_prim_infeas_cer:
128     print("Primal or dual infeasibility.\n")

```

```

129 elif mosek.solsta.unknown:
130     print("Unknown solution status")
131 else:
132     print("Other solution status")
133
134 # call the main function
135 try:
136     main()
137 except mosek.Exception, e:
138     print "ERROR: %s" % str(e.errno)
139     if e.msg is not None:
140         import traceback
141         traceback.print_exc()
142         print "\t%s" % e.msg
143     sys.exit(1)
144 except:
145     import traceback
146     traceback.print_exc()
147     sys.exit(1)
148 print "Finished OK"
149 sys.exit(0)

```

5.3.1.2 Example code comments

Most of the functionality in this example has already been explained for the linear optimization example in Section 5.2 and it will not be repeated here.

This example introduces one new function, `Task.putqobj`, which is used to input the quadratic terms of the objective function.

Since Q^o is symmetric only the lower triangular part of Q^o is inputted. The upper part of Q^o is computed by MOSEK using the relation

$$Q_{ij}^o = Q_{ji}^o.$$

Entries from the upper part may *not* appear in the input.

The lower triangular part of the matrix Q^o is specified using an unordered sparse triplet format (for details, see Section 5.8.3):

```

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

```

Please note that

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

Finally, the matrix Q^o is loaded into the task:

```

98 task.putqobj(qsubi, qsubj, qval)

```

5.3.2 Example: Quadratic constraints

In this section describes how to solve a problem with quadratic constraints. Please note that quadratic constraints are subject to the convexity requirement (5.9).

Consider the problem:

$$\begin{aligned} & \text{minimize} && x_1^2 + 0.1x_2^2 + x_3^2 - x_1x_3 - x_2 \\ & \text{subject to} && 1 \leq x_1 + x_2 + x_3 - x_1^2 - x_2^2 - 0.1x_3^2 + 0.2x_1x_3, \\ & && x \geq 0. \end{aligned} \quad (5.14)$$

This is equivalent to

$$\begin{aligned} & \text{minimize} && 1/2x^T Q^o x + c^T x \\ & \text{subject to} && 1/2x^T Q^0 x + Ax \geq b, \end{aligned} \quad (5.15)$$

where

$$Q^o = \begin{bmatrix} 2 & 0 & -1 \\ 0 & 0.2 & 0 \\ -1 & 0 & 2 \end{bmatrix}, \quad c = \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix}, \quad A = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}, \quad b = 1. \quad (5.16)$$

$$Q^0 = \begin{bmatrix} -2 & 0 & 0.2 \\ 0 & -2 & 0 \\ 0.2 & 0 & -0.2 \end{bmatrix}. \quad (5.17)$$

5.3.2.1 Source code

```

1 #
2 # Copyright: Copyright (c) 1998-2011 MOSEK ApS, Denmark. All rights reserved.
3 #
4 # File:    lo2.py
5 #
6 # Purpose: Demonstrates how to solve small linear
7 #           optimization problem using the MOSEK Python API.
8 ##
9
10 import sys
11
12 import mosek
13 # If numpy is installed, use that, otherwise use the
14 # Mosek's array module.
15 try:
16     from numpy import array,zeros,ones
17 except ImportError:
18     from mosek.array import array, zeros, ones
19
20 # Since the actual value of Infinity is ignores, we define it solely
21 # for symbolic purposes:
22 inf = 0.0
23
24 # Define a stream printer to grab output from MOSEK
25 def streamprinter(text):
26     sys.stdout.write(text)

```

```

27     sys.stdout.flush()
28
29 # We might write everything directly as a script, but it looks nicer
30 # to create a function.
31 def main ():
32     # Make a MOSEK environment
33     env = mosek.Env ()
34     # Attach a printer to the environment
35     env.set_Stream (mosek.streamtype.log, streamprinter)
36
37     # Create a task
38     task = env.Task(0,0)
39     # Attach a printer to the task
40     task.set_Stream (mosek.streamtype.log, streamprinter)
41
42
43     # Set up and input bounds and linear coefficients
44     bkc = [ mosek.boundkey.lo ]
45     blc = [ 1.0 ]
46     buc = [ inf ]
47
48     bkc = [ mosek.boundkey.lo,
49             mosek.boundkey.lo,
50             mosek.boundkey.lo ]
51     blx = [ 0.0, 0.0, 0.0 ]
52     bux = [ inf, inf, inf ]
53
54     c = [ 0.0, -1.0, 0.0 ]
55
56     asub = [ array([0]), array([0]), array([0]) ]
57     aval = [ array([1.0]), array([1.0]), array([1.0]) ]
58
59     NUMVAR = len(bkc)
60     NUMCON = len(bkc)
61     NUMANZ = 3
62     # Give MOSEK an estimate of the size of the input data.
63     # This is done to increase the speed of inputting data.
64     # However, it is optional.
65     task.putmaxnumvar(NUMVAR)
66     task.putmaxnumcon(NUMCON)
67     task.putmaxnumanz(NUMANZ)
68     # Append 'NUMCON' empty constraints.
69     # The constraints will initially have no bounds.
70     task.append(mosek.accmode.con, NUMCON)
71
72     #Append 'NUMVAR' variables.
73     # The variables will initially be fixed at zero (x=0).
74     task.append(mosek.accmode.var, NUMVAR)
75
76     #Optionally add a constant term to the objective.
77     task.putcfix(0.0)
78
79     for j in range(NUMVAR):
80         # Set the linear term c_j in the objective.
81         task.putcj(j, c[j])
82         # Set the bounds on variable j
83         # blx[j] <= x_j <= bux[j]
84         task.putbound(mosek.accmode.var, j, bkc[j], blx[j], bux[j])

```

```

85     # Input column j of A
86     task.putavec(mosek.accmode.var, # Input columns of A.
87                 j,                 # Variable (column) index.
88                 asub[j],           # Row index of non-zeros in column j.
89                 aval[j])           # Non-zero Values of column j.
90
91     for i in range(NUMCON):
92         task.putbound(mosek.accmode.con,i,bkc[i],blc[i],buc[i])
93
94     # Set up and input quadratic objective
95
96     qsubi = [ 0, 1, 2, 2 ]
97     qsubj = [ 0, 1, 0, 2 ]
98     qval = [ 2.0, 0.2, -1.0, 2.0 ]
99
100    task.putqobj(qsubi,qsubj,qval)
101
102    # The lower triangular part of the Q^0
103    # matrix in the first constraint is specified.
104    # This corresponds to adding the term
105    # - x0^2 - x1^2 - 0.1 x2^2 + 0.2 x0 x2
106
107    qsubi = [ 0, 1, 2, 2 ]
108    qsubj = [ 0, 1, 2, 0 ]
109    qval = [ -2.0, -2.0, -0.2, 0.2 ]
110
111    # put Q^0 in constraint with index 0.
112
113    task.putqconk (0, qsubi,qsubj, qval);
114
115    # Input the objective sense (minimize/maximize)
116    task.putobjsense(mosek.objsense.minimize)
117
118    # Optimize the task
119    task.optimize()
120
121    # Print a summary containing information
122    # about the solution for debugging purposes
123    task.solutionsummary(mosek.streamtype.msg)
124
125    prosta = []
126    solsta = []
127    [prosta,solsta] = task.getsolutionstatus(mosek.soltype.itr)
128
129    # Output a solution
130    xx = zeros(NUMVAR, float)
131    task.getsolutionslice(mosek.soltype.itr,
132                          mosek.solitem.xx,
133                          0,NUMVAR,
134                          xx)
135
136    if solsta == mosek.solsta.optimal or solsta == mosek.solsta.near_optimal:
137        print("Optimal solution: %s" % xx)
138    elif solsta == mosek.solsta.dual_infeas_cer:
139        print("Primal or dual infeasibility.\n")
140    elif solsta == mosek.solsta.prim_infeas_cer:
141        print("Primal or dual infeasibility.\n")
142    elif solsta == mosek.solsta.near_dual_infeas_cer:

```

```

143     print("Primal or dual infeasibility.\n")
144     elif solsta == mosek.solsta.near_prim_infeas_cer:
145         print("Primal or dual infeasibility.\n")
146     elif mosek.solsta.unknown:
147         print("Unknown solution status")
148     else:
149         print("Other solution status")
150
151 # call the main function
152 try:
153     main ()
154 except mosek.Exception, (code,msg):
155     print "ERROR: %s" % str(code)
156     if msg is not None:
157         print "\t%s" % msg
158         sys.exit(1)
159 except:
160     import traceback
161     traceback.print_exc()
162     sys.exit(1)
163 sys.exit(0)

```

The only new function introduced in this example is `Task.putqconk`, which is used to add quadratic terms to the constraints. While `Task.putqconk` add quadratic terms to a specific constraint, it is also possible to input all quadratic terms in all constraints in one chunk using the `Task.putqcon` function.

5.4 Conic optimization

Conic problems are a generalization of linear problems, allowing constraints of the type

$$x \in \mathcal{C}$$

where \mathcal{C} is a convex cone.

MOSEK can solve conic optimization problems of the following form

$$\begin{aligned}
 & \text{minimize} && c^T x + c^f \\
 & \text{subject to} && l^c \leq Ax \leq u^c, \\
 & && l^x \leq x \leq u^x, \\
 & && x \in \mathcal{C}
 \end{aligned} \tag{5.18}$$

where \mathcal{C} is a cone. \mathcal{C} can be a product of cones, i.e.

$$\mathcal{C} = \mathcal{C}_0 \times \cdots \times \mathcal{C}_{p-1}$$

in which case $x \in \mathcal{C}$ means $x^t \in \mathcal{C}_t \subseteq \mathbb{R}^{n_t}$. Please note that the set of real numbers \mathbb{R} is itself a cone, so linear variables are still allowed.

MOSEK supports two specific cones apart from the real numbers:

- The quadratic cone:

$$\mathcal{C}_t = \left\{ x \in \mathbb{R}^{n_t} : x_1 \geq \sqrt{\sum_{j=2}^{n_t} x_j^2} \right\}.$$

- The rotated quadratic cone:

$$\mathcal{C}_t = \left\{ x \in \mathbb{R}^{n_t} : 2x_1x_2 \geq \sum_{j=3}^{n_t} x_j^2, x_1, x_2 \geq 0 \right\}.$$

When creating a conic problem in MOSEK, each cone is defined by a *cone type* (quadratic or rotated quadratic cone) and a list of variable indexes. To summarize:

- In MOSEK all variables belong to the set \mathbb{R} of reals, unless they are explicitly declared as belonging to a cone.
- Each variable may belong to one cone *at most*.

5.4.1 Example: cqo1

The problem

$$\begin{aligned} & \text{minimize} && x_4 + x_5 \\ & \text{subject to} && x_0 + x_1 + x_2 + x_3 = 1, \\ & && x_0, x_1, x_2, x_3 \geq 0, \\ & && x_4 \geq \sqrt{x_0^2 + x_2^2}, \\ & && x_5 \geq \sqrt{x_1^2 + x_3^2} \end{aligned} \tag{5.19}$$

is an example of a conic quadratic optimization problem. The problem includes a set of linear constraints and two quadratic cones.

5.4.1.1 Source code

```

1 #
2 # Copyright: Copyright (c) 1998-2011 MOSEK ApS, Denmark. All rights reserved.
3 #
4 # File:    lo2.py
5 #
6 # Purpose: Demonstrates how to solve small linear
7 #           optimization problem using the MOSEK Python API.
8 ##
9
10 import sys
11
12 import mosek
13 # If numpy is installed, use that, otherwise use the
14 # Mosek's array module.
15 try:
16     from numpy import array, zeros, ones
17 except ImportError:
18     from mosek.array import array, zeros, ones
19
20 # Since the actual value of Infinity is ignored, we define it solely
21 # for symbolic purposes:
22 inf = 0.0

```

```

23
24 # Define a stream printer to grab output from MOSEK
25 def streamprinter(text):
26     sys.stdout.write(text)
27     sys.stdout.flush()
28
29 # We might write everything directly as a script, but it looks nicer
30 # to create a function.
31 def main ():
32     # Make a MOSEK environment
33     env = mosek.Env ()
34     # Attach a printer to the environment
35     env.set_Stream (mosek.streamtype.log, streamprinter)
36
37     # Create a task
38     task = env.Task(0,0)
39     # Attach a printer to the task
40     task.set_Stream (mosek.streamtype.log, streamprinter)
41
42     bkc = [ mosek.boundkey.fx ]
43     blc = [ 1.0 ]
44     buc = [ 1.0 ]
45
46     c = [
47         0.0,      0.0,      0.0,      0.0,
48         0.0,      1.0,      1.0 ]
49     bkc = [ mosek.boundkey.lo, mosek.boundkey.lo, mosek.boundkey.lo,
50             mosek.boundkey.lo, mosek.boundkey.fr, mosek.boundkey.fr ]
51     blx = [
52         0.0,      0.0,      0.0,
53         0.0,      -inf,     -inf ]
54     bux = [
55         inf,      inf,      inf,
56         inf,      inf,      inf ]
57
58     asub = [ array([0]), array([0]), array([0]), array([0]) ]
59     aval = [ array([1.0]), array([1.0]), array([1.0]), array([1.0]) ]
60
61     NUMVAR = len(bkc)
62     NUMCON = len(bkc)
63     NUMANZ = 4
64     # Give MOSEK an estimate of the size of the input data.
65     # This is done to increase the speed of inputting data.
66     # However, it is optional.
67     task.putmaxnumvar(NUMVAR)
68     task.putmaxnumcon(NUMCON)
69     task.putmaxnumanz(NUMANZ)
70     # Append 'NUMCON' empty constraints.
71     # The constraints will initially have no bounds.
72     task.append(mosek.acemode.con, NUMCON)
73
74     #Append 'NUMVAR' variables.
75     # The variables will initially be fixed at zero (x=0).
76     task.append(mosek.acemode.var, NUMVAR)
77
78     #Optionally add a constant term to the objective.
79     task.putcfix(0.0)
80
81     for j in range(NUMVAR):
82         # Set the linear term c_j in the objective.

```

```

81     task.putcj(j,c[j])
82     # Set the bounds on variable j
83     # blx[j] <= x_j <= bux[j]
84     task.putbound(mosek.accmode.var,j,bkx[j],blx[j],bux[j])
85
86 for j in range(len(aval)):
87     # Input column j of A
88     task.putavec(mosek.accmode.var, # Input columns of A.
89                 j,                 # Variable (column) index.
90                 asub[j],           # Row index of non-zeros in column j.
91                 aval[j])           # Non-zero Values of column j.
92 for i in range(NUMCON):
93     task.putbound(mosek.accmode.con,i,bkc[i],blc[i],buc[i])
94
95 # Input the cones
96 task.appendcone(mosek.conetype.quad,
97                0.0,
98                [ 4, 0, 2 ])
99 task.appendcone(mosek.conetype.quad,
100                0.0,
101                [ 5, 1, 3 ])
102
103 # Input the objective sense (minimize/maximize)
104 task.putobjsense(mosek.objsense.minimize)
105
106 # Optimize the task
107 task.optimize()
108 # Print a summary containing information
109 # about the solution for debugging purposes
110 task.solutionsummary(mosek.streamtype.msg)
111 prosta = []
112 solsta = []
113 [prosta,solsta] = task.getsolutionstatus(mosek.soltype.itr)
114
115 # Output a solution
116 xx = zeros(NUMVAR, float)
117 task.getsolutionslice(mosek.soltype.itr,
118                       mosek.solitem.xx,
119                       0,NUMVAR,
120                       xx)
121
122 if solsta == mosek.solsta.optimal or solsta == mosek.solsta.near_optimal:
123     print("Optimal solution: %s" % xx)
124 elif solsta == mosek.solsta.dual_infeas_cer:
125     print("Primal or dual infeasibility.\n")
126 elif solsta == mosek.solsta.prim_infeas_cer:
127     print("Primal or dual infeasibility.\n")
128 elif solsta == mosek.solsta.near_dual_infeas_cer:
129     print("Primal or dual infeasibility.\n")
130 elif solsta == mosek.solsta.near_prim_infeas_cer:
131     print("Primal or dual infeasibility.\n")
132 elif mosek.solsta.unknown:
133     print("Unknown solution status")
134 else:
135     print("Other solution status")
136
137
138 # call the main function

```

```

139 try:
140     main ()
141 except mosek.Exception, (code,msg):
142     print "ERROR: %s" % str(code)
143     if msg is not None:
144         print "\t%s" % msg
145         sys.exit(1)
146 except:
147     import traceback
148     traceback.print_exc()
149     sys.exit(1)
150 sys.exit(0)

```

5.4.1.2 Source code comments

The only new function introduced in the example is `Task.appendcone`, which is called here:

```

96 task.appendcone(mosek.conetype.quad,
97                0.0,
98                [ 4, 0, 2 ])

```

Here `mosek.conetype.quad` defines the cone type, in this case it is a *quadratic cone*. The cone parameter 0.0 is currently not used by MOSEK — simply passing 0.0 will work.

The last argument is a list of indexes of the variables in the cone.

5.5 Integer optimization

An optimization problem where one or more of the variables are constrained to integer values is denoted an integer optimization problem.

5.5.1 Example: milo1

In this section the example

$$\begin{aligned}
 &\text{maximize} && x_0 + 0.64x_1 \\
 &\text{subject to} && 50x_0 + 31x_1 \leq 250, \\
 & && 3x_0 - 2x_1 \geq -4, \\
 & && x_0, x_1 \geq 0 \quad \text{and integer}
 \end{aligned} \tag{5.20}$$

is used to demonstrate how to solve a problem with integer variables.

5.5.1.1 Source code

The example (5.20) is almost identical to a linear optimization problem except for some variables being integer constrained. Therefore, only the specification of the integer constraints requires something new

compared to the linear optimization problem discussed previously. In MOSEK these constraints are specified using the function `Task.putvartype` as shown in the code:

```

94 task.putvartypelist([ 0, 1 ],
95                     [ mosek.variabletype.type_int,
96                     mosek.variabletype.type_int ])

```

The complete source for the example is listed below.

```

1  ##
2  #   Copyright: Copyright (c) 1998-2011 MOSEK ApS, Denmark. All rights reserved.
3  #
4  #   File:      mio1.py
5  #
6  #   Purpose:   Demonstrates how to solve a small mixed
7  #             integer linear optimization problem using the MOSEK Python API.
8  ##
9
10 import sys
11
12 import mosek
13 # If numpy is installed, use that, otherwise use the
14 # Mosek's array module.
15 try:
16     from numpy import array,zeros,ones
17 except ImportError:
18     from mosek.array import array, zeros, ones
19
20 # Since the actual value of Infinity is ignored, we define it solely
21 # for symbolic purposes:
22 inf = 0.0
23
24 # Define a stream printer to grab output from MOSEK
25 def streamprinter(text):
26     sys.stdout.write(text)
27     sys.stdout.flush()
28
29 # We might write everything directly as a script, but it looks nicer
30 # to create a function.
31 def main():
32     # Make a MOSEK environment
33     env = mosek.Env()
34     # Attach a printer to the environment
35     env.set_Stream(mosek.streamtype.log, streamprinter)
36
37     # Create a task
38     task = env.Task(0,0)
39     # Attach a printer to the task
40     task.set_Stream(mosek.streamtype.log, streamprinter)
41
42     bkc = [ mosek.boundkey.up, mosek.boundkey.lo ]
43     blc = [ -inf, -4.0 ]
44     buc = [ 250.0, inf ]
45
46     bkx = [ mosek.boundkey.lo, mosek.boundkey.lo ]
47     blx = [ 0.0, 0.0 ]
48     bux = [ inf, inf ]
49

```

```

50     c      = [          1.0,          0.64 ]
51
52     asub = [  array([0,  1]),    array([0,  1])  ]
53     aval = [  array([50.0, 3.0]), array([31.0, -2.0]) ]
54
55     NUMVAR = len(bkx)
56     NUMCON = len(bkc)
57     NUMANZ = 9
58     # Give MOSEK an estimate of the size of the input data.
59     # This is done to increase the speed of inputting data.
60     # However, it is optional.
61     task.putmaxnumvar(NUMVAR)
62     task.putmaxnumcon(NUMCON)
63     task.putmaxnumanz(NUMANZ)
64     # Append 'NUMCON' empty constraints.
65     # The constraints will initially have no bounds.
66     task.append(mosek.acemode.con, NUMCON)
67
68     #Append 'NUMVAR' variables.
69     # The variables will initially be fixed at zero (x=0).
70     task.append(mosek.acemode.var, NUMVAR)
71
72     # Optionally add a constant term to the objective.
73     task.putcfix(0.0)
74
75     for j in range(NUMVAR):
76         # Set the linear term c_j in the objective.
77         task.putcj(j, c[j])
78         # Set the bounds on variable j
79         # blx[j] <= x_j <= bux[j]
80         task.putbound(mosek.acemode.var, j, bkx[j], blx[j], bux[j])
81         # Input column j of A
82         task.putavecmosek(mosek.acemode.var, # Input columns of A.
83                           j,                # Variable (column) index.
84                           asub[j],          # Row index of non-zeros in column j.
85                           aval[j])          # Non-zero Values of column j.
86
87     for i in range(NUMCON):
88         task.putbound(mosek.acemode.con, i, bkc[i], blc[i], buc[i])
89
90     # Input the objective sense (minimize/maximize)
91     task.putobjsense(mosek.objsense.maximize)
92
93     # Define variables to be integers
94     task.putvartypelist([ 0, 1 ],
95                          [ mosek.variabletype.type_int,
96                            mosek.variabletype.type_int ])
97
98     # Optimize the task
99     task.optimize()
100
101     # Print a summary containing information
102     # about the solution for debugging purposes
103     task.solutionsummary(mosek.streamtype.msg)
104
105     prosta = []
106     solsta = []
107     [prosta, solsta] = task.getsolutionstatus(mosek.soltype.itg)

```

```

108
109 # Output a solution
110 xx = zeros(NUMVAR, float)
111 task.getsolutionslice(mosek.soltype.itg,
112                      mosek.solitem.xx,
113                      0, NUMVAR,
114                      xx)
115
116 if solsta == mosek.solsta.optimal or solsta == mosek.solsta.near_optimal:
117     print("Optimal solution: %s" % xx)
118 elif solsta == mosek.solsta.dual_infeas_cer:
119     print("Primal or dual infeasibility.\n")
120 elif solsta == mosek.solsta.prim_infeas_cer:
121     print("Primal or dual infeasibility.\n")
122 elif solsta == mosek.solsta.near_dual_infeas_cer:
123     print("Primal or dual infeasibility.\n")
124 elif solsta == mosek.solsta.near_prim_infeas_cer:
125     print("Primal or dual infeasibility.\n")
126 elif mosek.solsta.unknown:
127     print("Unknown solution status")
128 else:
129     print("Other solution status")
130
131 # call the main function
132 try:
133     main ()
134 except mosek.Exception, (code,msg):
135     print "ERROR: %s" % str(code)
136     if msg is not None:
137         print "\t%s" % msg
138         sys.exit(1)
139 except:
140     import traceback
141     traceback.print_exc()
142     sys.exit(1)
143 sys.exit(0)

```

5.5.1.2 Code comments

Please note that when `Task.getsolutionslice` is called, the integer solution is requested by using `mosek.soltype.itg`. No dual solution is defined for integer optimization problems.

5.5.2 Specifying an initial solution

Integer optimization problems are generally hard to solve, but the solution time can often be reduced by providing an initial solution for the solver. Solution values can be set using `Task.putsolution` (for inputting a whole solution) or `Task.putsolutioni` (for inputting solution values related to a single variable or constraint).

It is not necessary to specify the whole solution. By setting the `mosek.iparam.mio_construct_sol` parameter to `mosek.onoffkey.on` and inputting values for the integer variables only, will force MOSEK to compute the remaining continuous variable values.

If the specified integer solution is infeasible or incomplete, MOSEK will simply ignore it.

5.5.3 Example: Specifying an integer solution

Consider the problem

$$\begin{aligned} &\text{maximize} && 7x_0 + 10x_1 + x_2 + 5x_3 \\ &\text{subject to} && x_0 + x_1 + x_2 + x_3 \leq 2.5 \\ &&& x_0, x_1, x_2 \text{ integer, } x_0, x_1, x_2, x_3 \geq 0 \end{aligned} \tag{5.21}$$

The following example demonstrates how to optimize the problem using a feasible starting solution generated by selecting the integer values as $x_0 = 0, x_1 = 2, x_2 = 0$.

```

1  ##
2  #   Copyright: Copyright (c) 1998-2011 MOSEK ApS, Denmark. All rights reserved.
3  #
4  #   File:      mioinitsol.cs
5  #
6  #   Purpose:   Demonstrates how to solve a small mixed
7  #               integer linear optimization problem using the MOSEK Python API.
8  ##
9
10 import sys
11
12 import mosek
13
14 # If numpy is installed, use that, otherwise use the
15 # Mosek's array module.
16 try:
17     from numpy import array,zeros,ones
18 except ImportError:
19     from mosek.array import array, zeros, ones
20
21 # Since the actual value of Infinity is ignored, we define it solely
22 # for symbolic purposes:
23 inf = 0.0
24
25
26 # Define a stream printer to grab output from MOSEK
27 def streamprinter(text):
28     sys.stdout.write(text)
29     sys.stdout.flush()
30
31
32 # We might write everything directly as a script, but it looks nicer
33 # to create a function.
34 def main():
35     # Make a MOSEK environment
36     env = mosek.Env()
37     # Attach a printer to the environment
38     env.set_Stream(mosek.streamtype.log, streamprinter)
39
40     # Create a task
41     task = env.Task(0,0)
42     # Attach a printer to the task

```

```

43 task.set_Stream (mosek.streamtype.log, streamprinter)
44
45
46 bkc = [ mosek.boundkey.up ]
47 blc = [ -inf,                ]
48 buc = [ 2.5                  ]
49
50 bkc = [ mosek.boundkey.lo,
51         mosek.boundkey.lo,
52         mosek.boundkey.lo,
53         mosek.boundkey.lo ]
54
55 blx = [0.0, 0.0, 0.0, 0.0 ]
56 bux = [ inf, inf,  inf,  inf ]
57
58 c   = [ 7.0, 10.0, 1.0, 5.0 ]
59
60 asub = [ 0, 0, 0, 0 ]
61 acof = [ 1.0, 1.0, 1.0, 1.0]
62
63 ptrb = [ 0, 1, 2, 3 ]
64 ptre = [ 1, 2, 3, 4 ]
65
66 numvar = len(bkc)
67 numcon = len(bkc)
68
69 # Input linear data
70 task.inputdata(numcon,numvar,
71               c,0.0,
72               ptrb, ptre, asub, acof,
73               bkc,  blc,  buc,
74               bkc,  blx,  bux)
75
76 # Input objective sense
77 task.putobjsense(mosek.objsense.maximize)
78
79 # Define variables to be integers
80 task.putvartypelist([ 0, 1, 2 ],
81                    [ mosek.variabletype.type_int,
82                      mosek.variabletype.type_int,
83                      mosek.variabletype.type_int])
84
85 # Construct an initial feasible solution from the
86 #   values of the integer valuse specified
87 task.putintparam(mosek.iparam.mio_construct_sol,
88                 mosek.onoffkey.on);
89
90 # Set status of all variables to unknown
91 task.makesolutionstatusunknown(mosek.soltype.itg);
92
93 # Assign values 1,1,0 to integer variables
94 task.putsolutioni (
95     mosek.accmode.var,
96     0,
97     mosek.soltype.itg,
98     mosek.stakey.supbas,
99     0.0,
100    0.0,

```

```

101         0.0,
102         0.0);
103
104     task.putsolutioni (
105         mosek.accmode.var,
106         1,
107         mosek.soltype.itg,
108         mosek.stakey.supbas,
109         2.0,
110         0.0,
111         0.0,
112         0.0);
113
114
115     task.putsolutioni (
116         mosek.accmode.var,
117         2,
118         mosek.soltype.itg,
119         mosek.stakey.supbas,
120         0.0,
121         0.0,
122         0.0,
123         0.0);
124
125     # Optimize
126     task.optimize()
127
128     if task.solutiondef(mosek.soltype.itg):
129
130         # Output a solution
131         xx = zeros(numvar, float)
132         task.getsolutionslice(mosek.soltype.itg,
133                               mosek.solitem.xx,
134                               0, numvar,
135                               xx)
136         print "x =", xx
137     else:
138         print "Integer solution not defined. Probably a problem with 'mosekglob' optimizer."
139
140 # call the main function
141 try:
142     main ()
143 except mosek.Exception, e:
144     print "ERROR: %s" % str(e.errno)
145     if e.msg is not None:
146         print "\t%s" % e.msg
147     sys.exit(1)
148 except:
149     import traceback
150     traceback.print_exc()
151     sys.exit(1)

```

5.6 Problem modification and reoptimization

Often one might want to solve not just a single optimization problem, but a sequence of problem, each differing only slightly from the previous one. This section demonstrates how to modify and reoptimize an existing problem. The example we study is a simple production planning model.

5.6.1 A production planning problem

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

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

With the current resources available, the company has 100,000 minutes of assembly time, 50,000 minutes of polishing time and 60,000 minutes of packing time available per year.

Now the question is how many items of each product the company should produce each year in order to maximize profit?

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

$$\begin{aligned}
 & \text{maximize} && 1.5x_0 + 2.5x_1 + 3.0x_2 \\
 & \text{subject to} && 2x_0 + 4x_1 + 3x_2 \leq 100000, \\
 & && 3x_0 + 2x_1 + 3x_2 \leq 50000, \\
 & && 2x_0 + 3x_1 + 2x_2 \leq 60000,
 \end{aligned} \tag{5.22}$$

and

$$x_0, x_1, x_2 \geq 0. \tag{5.23}$$

The following code loads this problem into the optimization task.

```

28 # Create a MOSEK environment
29 env = mosek.Env ()
30 # Attach a printer to the environment
31 env.set_Stream (mosek.streamtype.log, streamprinter)
32 # Initialize the environment
33 env.init ()
34
35 # Create a task
36 task = env.Task(0,0)
37 # Attach a printer to the task
38 task.set_Stream (mosek.streamtype.log, streamprinter)
39
40 # Bound keys for constraints
41 bkc = [ mosek.boundkey.up,
42         mosek.boundkey.up,

```

```

43     mosek.boundkey.up]
44 # Bound values for constraints
45 blc = array ([-inf, -inf, -inf])
46 buc = array ([100000.0 , 50000.0, 60000.0])
47
48 # Bound keys for variables
49 bkc = [ mosek.boundkey.lo,
50         mosek.boundkey.lo,
51         mosek.boundkey.lo ]
52 # Bound values for variables
53 blx = array ([ 0.0,  0.0,  0.0])
54 bux = array ([+inf, +inf, +inf])
55
56 # Objective coefficients
57 csub = array([  0,  1,  2 ])
58 cval = array([ 1.5, 2.5, 3.0 ])
59
60 # We input the A matrix column-wise
61 # asub contains row indexes
62 asub = array([ 0, 1, 2,
63               0, 1, 2,
64               0, 1, 2])
65 # acof contains coefficients
66 acof = array([ 2.0, 3.0, 2.0,
67               4.0, 2.0, 3.0,
68               3.0, 3.0, 2.0 ])
69 # aptrb and aptre contains the offsets into asub and acof where
70 # columns start and end respectively
71 aptrb = array([ 0, 3, 6 ])
72 aptre = array([ 3, 6, 9 ])
73
74 numvar = len(bkc)
75 numcon = len(bkc)
76
77 # Append the constraints
78 task.append(mosek.accmode.con, numcon)
79
80 # Append the variables.
81 task.append(mosek.accmode.var, numvar)
82
83 # Input objective
84 task.putcfix(0.0)
85 task.putclist(csub, cval)
86
87 # Put constraint bounds
88 task.putboundslice(mosek.accmode.con,
89                    0, numcon,
90                    bkc, blc, buc)
91
92 # Put variable bounds
93 task.putboundslice(mosek.accmode.var,
94                    0, numvar,
95                    bkc, blx, bux)
96
97
98
99 # Input A non-zeros by columns
100 for j in range(numvar):

```

```

101     ptrb,ptre = aptrb[j],aptre[j]
102     task.putavec(mosek.accmode.var,j,
103                 asub[ptrb:ptre],
104                 acof[ptrb:ptre])
105
106 # Input the objective sense (minimize/maximize)
107 task.putobjsense(mosek.objsense.maximize)
108
109
110 # Optimize the task
111 task.optimize()
112
113 # Output a solution
114 xx = zeros(numvar, float)
115 task.getsolutionslice(mosek.soltype.bas,
116                       mosek.solitem.xx,
117                       0,numvar,
118                       xx)
119 print "xx =", xx

```

5.6.2 Changing the A matrix

Suppose we want to change the time required for assembly of product 0 to 3 minutes. This corresponds to setting $a_{0,0} = 3$, which is done by calling the function `Task.putaij` as shown below.

```

123 task.putaij(0, 0, 3.0)

```

The problem now has the form:

$$\begin{aligned}
 &\text{maximize} && 1.5x_0 &+& 2.5x_1 &+& 3.0x_2 \\
 &\text{subject to} && 3x_0 &+& 4x_1 &+& 3x_2 &\leq 100000, \\
 & && 3x_0 &+& 2x_1 &+& 3x_2 &\leq 50000, \\
 & && 2x_0 &+& 3x_1 &+& 2x_2 &\leq 60000,
 \end{aligned} \tag{5.24}$$

and

$$x_0, x_1, x_2 \geq 0. \tag{5.25}$$

After changing the A matrix we can find the new optimal solution by calling

`Task.optimize`

again

5.6.3 Appending variables

We now want to add a new product with the following data:

Product no.	Assembly (minutes)	Polishing (minutes)	Packing (minutes)	Profit (\$)
3	4	0	1	1.00

This corresponds to creating a new variable x_3 , appending a new column to the A matrix and setting a new value in the objective. We do this in the following code.

```

125 # Append a new variable x_3 to the problem */
126 task.append(mosek.accmode.var,1)
127
128 # Set bounds on new variable
129 task.putbound(mosek.accmode.var,
130               task.getnumvar()-1,
131               mosek.boundkey.lo,
132               0,
133               +inf)
134
135 # Change objective
136 task.putcj(task.getnumvar()-1,1.0)
137
138 # Put new values in the A matrix
139 acolsub = array([0, 2])
140 acolval = array([4.0, 1.0])
141
142 task.putavec(mosek.accmode.var,
143              task.getnumvar()-1, # column index
144              acolsub,
145              acolval)

```

After this operation the problem looks this way:

$$\begin{aligned}
& \text{maximize} && 1.5x_0 + 2.5x_1 + 3.0x_2 + 1.0x_3 \\
& \text{subject to} && 3x_0 + 4x_1 + 3x_2 + 4x_3 \leq 100000, \\
& && 3x_0 + 2x_1 + 3x_2 \leq 50000, \\
& && 2x_0 + 3x_1 + 2x_2 + 1x_3 \leq 60000,
\end{aligned} \tag{5.26}$$

and

$$x_0, x_1, x_2, x_3 \geq 0. \tag{5.27}$$

5.6.4 Reoptimization

When

`Task.optimize`

is called MOSEK will store the optimal solution internally. After a task has been modified and

`Task.optimize`

is called again the solution will automatically be used to reduce solution time of the new problem, if possible.

In this case an optimal solution to problem (5.24) was found and then added a column was added to get (5.26). The simplex optimizer is well suited for exploiting an existing primal or dual feasible solution. Hence, the subsequent code instructs MOSEK to choose the simplex optimizer freely when optimizing.

```

146 # Change optimizer to simplex free and reoptimize

```

```

147 task.putintparam(mosek.iparam.optimizer,mosek.optimizertype.free_simplex)
148 task.optimize()

```

5.6.5 Appending constraints

Now suppose we want to add a new stage to the production 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

$$x_0 + 2x_1 + x_2 + x_3 \leq 30000 \quad (5.28)$$

to the problem which is done in the following code:

```

149 # Append a new constraint
150 task.append(mosek.accmode.con,1)
151
152 # Set bounds on new constraint
153 task.putbound(
154     mosek.accmode.con,
155     task.getnumcon()-1, # row index
156     mosek.boundkey.up,
157     -inf,
158     30000)
159
160 # Put new values in the A matrix
161
162 arowsub = array([0, 1, 2, 3 ])
163 arowval = array([1.0, 2.0, 1.0, 1.0])
164
165 task.putavec(mosek.accmode.con,
166     task.getnumcon()-1, # row index
167     arowsub,
168     arowval)

```

5.7 Efficiency considerations

Although MOSEK is implemented to handle memory efficiently, the user may have valuable knowledge about a problem, which could be used to improve the performance of MOSEK. This section discusses some tricks and general advice that hopefully make MOSEK process your problem faster.

Avoid memory fragmentation: MOSEK stores the optimization problem in internal data structures in the memory. Initially MOSEK will allocate structures of a certain size, and as more items are added to the problem the structures are reallocated. For large problems the same structures may be reallocated many times causing memory fragmentation. One way to avoid this is to give MOSEK an estimated size of your problem using the functions:

- `Task.putmaxnumvar`. Estimate for the number of variables.
- `Task.putmaxnumcon`. Estimate for the number of constraints.
- `Task.putmaxnumcone`. Estimate for the number of cones.
- `Task.putmaxnumanz64`. Estimate for the number of non-zeros in A .
- `Task.putmaxnumqnz64`. Estimate for the number of non-zeros in the quadratic terms.

None of these functions change the problem, they only give hints to the eventual dimension of the problem. If the problem ends up growing larger than this, the estimates are automatically increased.

Tune the reallocation process: It is possible to obtain information about how often MOSEK re-allocates storage for the A matrix by inspecting `mosek.iinfitem.sto.num.a.realloc`. A large value indicates that `maxnumanz` has been reestimated many times and that the initial estimate should be increased.

Do not mix put- and get- functions: For instance, the functions `Task.putavec` and `Task.getavec`. MOSEK will queue put- commands internally until a get- function is called. If every put- function call is followed by a get- function call, the queue will have to be flushed often, decreasing efficiency.

In general get- commands should not be called often during problem setup.

Use the LIFO principle when removing constraints and variables: MOSEK can more efficiently remove constraints and variables with a high index than a small index.

An alternative to removing a constraint or a variable is to fix it at 0, and set all relevant coefficients to 0. Generally this will not have any impact on the optimization speed.

Add more constraints and variables than you need (now): The cost of adding one constraint or one variable is about the same as adding many of them. Therefore, it may be worthwhile to add many variables instead of one. Initially fix the unused variable at zero, and then later unfix them as needed. Similarly, you can add multiple free constraints and then use them as needed.

Use one environment (env) only: If possible share the environment (`env`) between several tasks. For most applications you need to create only a single `env`.

Do not remove basic variables: When doing reoptimizations, instead of removing a basic variable it may be more efficient to fix the variable at zero and then remove it when the problem is reoptimized and it has left the basis. This makes it easier for MOSEK to restart the simplex optimizer.

5.7.1 API overhead

The Python interface is a thin wrapper around a native MOSEK library. The layer between the Python application and the native MOSEK library is made as thin as possible to minimize the overhead from function calls.

The methods in `mosek.Env` and `mosek.Task` are all written in C and resides in the module `pymosek`. Each method converts the call parameter data structures (i.e. creates a complete copy of the data), calls a MOSEK function and converts the returned values back into Python structures.

All data are copied *at least* once. For larger problems this may mean, that fetching or inputting large chunks of data is less expensive than fetching/inputting the same data as single values.

5.8 Conventions employed in the API

5.8.1 Naming conventions for arguments

In the definition of the MOSEK Python API 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 5.2 the variable names used to specify the problem parameters are listed.

The relation between the variable names and the problem parameters is as follows:

- The quadratic terms in the objective:

$$q_{qosubi[t],qosubj[t]}^o = qoval[t], \quad t = 0, \dots, numqonz - 1. \quad (5.29)$$

- The linear terms in the objective:

$$c_j = c[j], \quad j = 0, \dots, numvar - 1 \quad (5.30)$$

- The fixed term in the objective:

$$c^f = cfix. \quad (5.31)$$

- The quadratic terms in the constraints:

$$q_{qcsubi[t],qcsbj[t]}^{qcsbk[t]} = qcval[t], \quad t = 0, \dots, numqcnz - 1. \quad (5.32)$$

- The linear terms in the constraints:

$$a_{asub[t],j} = aval[t], \quad t = ptrb[j], \dots, ptre[j] - 1, \\ j = 0, \dots, numvar - 1. \quad (5.33)$$

- The bounds on the constraints are specified using the variables `bkc`, `blc`, and `buc`. The components of the integer array `bkc` specify the bound type according to Table 5.3. For instance

Python name	Python type	Dimension	Related problem parameter
<code>numcon</code>	<code>int</code>		m
<code>numvar</code>	<code>int</code>		n
<code>numcone</code>	<code>int</code>		t
<code>numqonz</code>	<code>int</code>		q_{ij}^o
<code>qosubi</code>	<code>int[]</code>	<code>numqonz</code>	q_{ij}^o
<code>qosubj</code>	<code>int[]</code>	<code>numqonz</code>	q_{ij}^o
<code>c</code>	<code>float[]</code>	<code>numvar</code>	c_j
<code>cfix</code>	<code>float</code>		c^f
<code>numqcnz</code>	<code>int</code>		q_{ij}^k
<code>qcsubk</code>	<code>int[]</code>	<code>qcnz</code>	q_{ij}^k
<code>qcsubi</code>	<code>int[]</code>	<code>qcnz</code>	q_{ij}^k
<code>qcsubj</code>	<code>int[]</code>	<code>qcnz</code>	q_{ij}^k
<code>aptrb</code>	<code>int[]</code>	<code>numvar</code>	a_{ij}
<code>aptre</code>	<code>int[]</code>	<code>numvar</code>	a_{ij}
<code>asub</code>	<code>int[]</code>	<code>aptre[numvar-1]</code>	a_{ij}
<code>aval</code>	<code>float[]</code>	<code>aptre[numvar-1]</code>	a_{ij}
<code>blc</code>	<code>float[]</code>	<code>numcon</code>	l_k^c
<code>buc</code>	<code>float[]</code>	<code>numcon</code>	u_k^c
<code>blx</code>	<code>float[]</code>	<code>numvar</code>	l_k^x
<code>bux</code>	<code>float[]</code>	<code>numvar</code>	u_k^x

Table 5.2: Naming convention used in MOSEK. Here `TTT[]` means a `Numeric.array` of type `TTT`.

Symbolic constant	Lower bound	Upper bound
<code>mosek.boundkey.fx</code>	finite	identical to the lower bound
<code>mosek.boundkey.fr</code>	minus infinity	plus infinity
<code>mosek.boundkey.lo</code>	finite	plus infinity
<code>mosek.boundkey.ra</code>	finite	finite
<code>mosek.boundkey.up</code>	minus infinity	finite

Table 5.3: Interpretation of the bound keys.

`bkc[2]=mosek.boundkey.lo` means that $-\infty < l_2^c$ and $u_2^c = \infty$. Finally, the numerical values of the bounds are given by

$$l_k^c = \text{blc}[k], \quad k = 0, \dots, \text{numcon} - 1 \quad (5.34)$$

and

$$u_k^c = \text{buc}[k], \quad k = 0, \dots, \text{numcon} - 1. \quad (5.35)$$

- The bounds on the variables are specified using the variables `bkx`, `blx`, and `bux`. The components in the integer array `bkx` specify the bound type according to Table 5.3. The numerical values for the lower bounds on the variables are given by

$$l_j^x = \text{blx}[j], \quad j = 0, \dots, \text{numvar} - 1. \quad (5.36)$$

The numerical values for the upper bounds on the variables are given by

$$u_j^x = \text{bux}[j], \quad j = 0, \dots, \text{numvar} - 1. \quad (5.37)$$

5.8.1.1 Bounds

A bound on a variable or on a constraint in MOSEK consists of a *bound key*, as defined in Table 5.3, a lower bound value and an upper bound value. Even if a variable or constraint is bounded only from below, e.g. $x \geq 0$, both bounds are inputted or extracted; the value inputted as upper bound for ($x \geq 0$) is ignored.

5.8.2 Vector formats

Three different vector formats are used in the MOSEK API:

Full 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`, one would write

```
28 c = zeros(numvar,float)
29 task.getc(c)
```

where `numvar` is the number of variables in the problem.

Vector slice: A vector slice is a range of values. For example, to get the bounds associated constraint 3 through 10 (both inclusive) one would write

```
31 upper_bound = zeros(8,float)
32 lower_bound = zeros(8,float)
33 bound_key   = array([None] * 8)
34
35 task.getboundslice(accmode.con, 2, 10,
36                     bound_key, lower_bound, upper_bound)
```

Please note that items in MOSEK are numbered from 0, so that the index of the first item is 0, and the index of the n 'th item is $n - 1$.

Sparse vector: A sparse vector is given as an array of indexes and an array of values. For example, to input a set of bounds associated with constraints number 1, 6, 3, and 9, one might write

```

38 bound_index = [      1,      6,      3,      9]
39 bound_key   = [boundkey.fr,boundkey.lo,boundkey.up,boundkey.fx]
40 lower_bound = [      0.0,     -10.0,      0.0,      5.0]
41 upper_bound = [      0.0,      0.0,      6.0,      5.0]
42 task.putboundlist(accmode.con, bound_index,
43                   bound_key,lower_bound,upper_bound)

```

Note that the list of indexes need not be ordered.

5.8.3 Matrix formats

The coefficient matrices in a problem are inputted and extracted in a sparse format, either as complete or a partial matrices. Basically there are two different formats for this.

5.8.3.1 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:

```

45 subi = array([ 1, 3, 5 ])
46 subj = array([ 2, 3, 4 ])
47 cof  = array([ 1.1, 4.3, 0.2 ])
48 task.putaijlist(subi,subj,cof)

```

Please note that in some cases (like `Task.putaijlist`) *only* the specified indexes remain 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.

5.8.3.2 Row or column ordered sparse matrix

In a sparse matrix format only the non-zero entries of the matrix are stored. MOSEK uses a sparse matrix format ordered either by rows or columns. In the column-wise format the position of the non-zeros are given as a list of row indexes. In the row-wise format the position of the non-zeros are given as a list of column indexes. Values of the non-zero entries are given in column or row order.

A sparse matrix in column ordered format consists of:

asub: List of row indexes.

aval: List of non-zero entries of A ordered by columns.

ptrb: Where `ptrb[j]` is the position of the first value/index in `aval` / `asub` for column j .

ptre: Where `ptre[j]` is the position of the last value/index plus one in `aval` / `asub` for column j .

The values of a matrix A with `numcol` columns are assigned so that for

$$j = 0, \dots, \text{numcol} - 1.$$

We define

$$a_{\text{asub}[k],j} = \text{aval}[k], \quad k = \text{ptrb}[j], \dots, \text{ptre}[j] - 1. \quad (5.38)$$

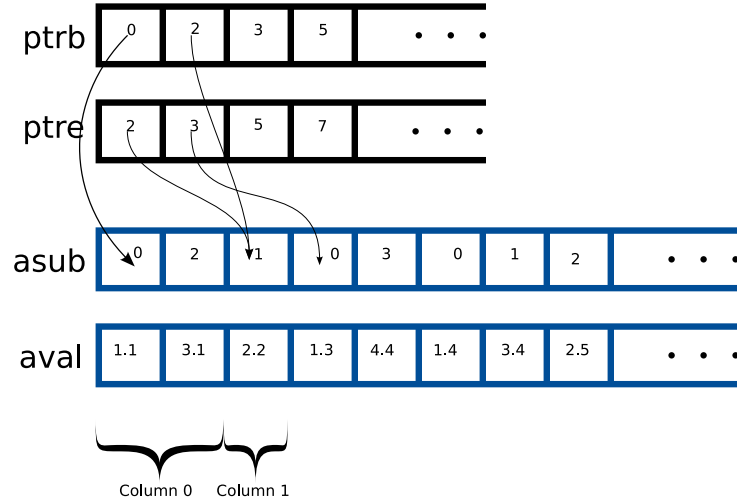


Figure 5.1: The matrix A (5.39) represented in column ordered sparse matrix format.

As an example consider the matrix

$$A = \begin{bmatrix} 1.1 & & 1.3 & 1.4 & \\ & 2.2 & & & 2.5 \\ 3.1 & & & 3.4 & \\ & & 4.4 & & \end{bmatrix}. \quad (5.39)$$

which can be represented in the column ordered sparse matrix format as

$$\begin{aligned} \text{ptrb} &= [0, 2, 3, 5, 7], \\ \text{ptre} &= [2, 3, 5, 7, 8], \\ \text{asub} &= [0, 2, 1, 0, 3, 0, 2, 1], \\ \text{aval} &= [1.1, 3.1, 2.2, 1.3, 4.4, 1.4, 3.4, 2.5]. \end{aligned}$$

Fig. 5.1 illustrates how the matrix A (5.39) is represented in column ordered sparse matrix format.

5.8.3.3 Row ordered sparse matrix

The matrix A (5.39) can also be represented in the row ordered sparse matrix format as:

```

ptrb = [0, 3, 5, 7],
ptre = [3, 5, 7, 8],
asub = [0, 2, 3, 1, 4, 0, 3, 2],
aval = [1.1, 1.3, 1.4, 2.2, 2.5, 3.1, 3.4, 4.4].

```

5.8.4 Array objects

The MOSEK Python API provides a simple array object in the module `mosekarr`. This includes a one-dimensional dense array which can be of type `Float`, `Int` or `Object`, and a few operators and functions to create and modify array objects.

Arrays can be constructed in several ways:

```

60 # Create an array of integers
61 a0 = array([1,2,3],int)
62 # Create an array of floats
63 a1 = array([1,2,3],float)
64 # Create an integer array of ones
65 a2 = ones(10)
66 # Create a float array of ones
67 a3 = ones(10,float)
68 # Create a range of integers 5,6,...,9
69 a4 = range(5,10)
70 # Create and array of objects
71 a5 = array(['a string', 'b string', 10, 2.2])

```

A limited set of operations on arrays are available - these should work more or less like the equivalent Numeric operations:

```

73 a = ones(10,float)
74 b = 1.0 * arange(10)
75
76
77 # element-wise multiplication, addition and subtraction
78 c0 = a * b
79 c1 = a + b
80 c2 = a - b
81
82 # multiplly each element by 2.1
83 c4 = a * 2.1
84
85 # add 2 to each element
86 c5 = a + 2

```

If more advanced array operations is needed, it is necessary to install the Python Numeric package.

5.8.5 Typical problems using the Python API

Since all type-information in Python is implicit, type-checking is performed only when required, and in certain cases it is necessary to explicitly write type information.

The MOSEK API currently *only* supports its own array object (`mosek.array.array`) and Python `numpy arrays`. Other array or list compatible objects will be accepted but are converted.

Typically type errors occur in two situations:

- An array argument did not have the right type and could not be converted.
- An array was expected, but the argument was not an array and not a list-compatible object.

Furthermore, please note that `mosek.array` module only supports a limited set of array types: `int32`, `int64`, `float64` and `bool`. The numerical types support normal simple mathematical operation (addition, subtraction, multiplication etc.)

5.9 The license system

By default a license token is checked out when `Task.optimize` is first called and is returned when the MOSEK environment is deleted. Calling `Task.optimize` from different threads using the same MOSEK environment only consumes one license token.

To change the license systems behavior to returning the license token after each call to `Task.optimize` set the parameter `mosek.iparam.cache_license` to `mosek.onoffkey.off`. Please note that there is a small overhead associated with setting this parameter, since checking out a license token from the license server can take a small amount of time.

Additionally license checkout and checkin can be controlled manually with the functions `Env.checkinlicense` and `Env.checkoutlicense`.

5.9.1 Waiting for a free license

By default an error will be returned if no license token is available. By setting the parameter `mosek.iparam.license_wait` MOSEK can be instructed to wait until a license token is available.

Chapter 6

Advanced API tutorial

This chapter provides information about additional problem classes and functionality provided in the Python API.

6.1 Linear network flow problems

Network flow problems are a special class of linear optimization problems which has many applications. A network consists of a set of points connected by a set of lines. Usually the points and lines are called *nodes* and *arcs*. Arcs may have an direction on them. The network is directed if all arcs are directed. The class of network flow problems is defined as follows.

Let $G = (\mathcal{N}, \mathcal{A})$ be a directed network of nodes \mathcal{N} and arcs \mathcal{A} . Associated with every arc $(i, j) \in \mathcal{A}$ is a cost c_{ij} and a capacity $[l_{ij}^x, u_{ij}^x]$. Moreover, associated with each node $i \in \mathcal{N}$ in the network is a lower limit l_i^c and an upper limit u_i^c on the demand (supply) of the node. The minimum cost of a network flow problem can be stated as follows:

$$\begin{aligned} & \text{minimize} && \sum_{(i,j) \in \mathcal{A}} c_{ij} x_{ij} \\ \text{subject to} & l_i^c &\leq \sum_{\{j: (i,j) \in \mathcal{A}\}} x_{ij} - \sum_{\{j: (j,i) \in \mathcal{A}\}} x_{ji} &\leq u_i^c \quad \forall i \in \mathcal{N}, \\ & l_{ij}^x &\leq x_{ij} &\leq u_{ij}^x \quad \forall (i,j) \in \mathcal{A}. \end{aligned} \tag{6.1}$$

A classical example of a network flow problem is the transportation problem where the objective is to distribute goods from warehouses to customers at lowest possible total cost, see [2] for a detailed application reference.

The above graph formulation of the network flow problem implies the structural properties. Each variable appears in exactly two constraints with a numerical value of either -1.0 or $+1.0$.

It is well-known that problems with network flow structure can be solved efficiently with a specialized version of the simplex method. MOSEK includes such a network simplex implementation which can be called either directly using `Task.netoptimize` or indirectly by letting the standard simplex optimizer extract the embedded network. This section shows how to solve a network problem by a direct call to

Task.netoptimize. For further details on how to exploit embedded network in the standard simplex optimizer, see Section 8.3.1.

6.1.1 A linear network flow problem example

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

$$\begin{array}{rcll}
 \text{maximize} & x_0 & + & x_2 & + & & - & x_4 & + & x_5 & & \\
 \text{subject to} & -x_0 & & & + & x_3 & & & & & & = & 1, \\
 & & & x_2 & - & x_3 & + & x_4 & + & x_5 & & = & -2, \\
 & x_0 & - & x_1 & & & & - & x_4 & - & x_5 & = & 0, \\
 & & x_1 & - & x_2 & + & & & & & & = & 0,
 \end{array} \tag{6.2}$$

having the bounds $0 \leq x_j \leq \infty$ for $j = 0 \dots 5$.

The corresponding graph $G = (\mathcal{N}, \mathcal{A})$ is displayed in fig.6.1.

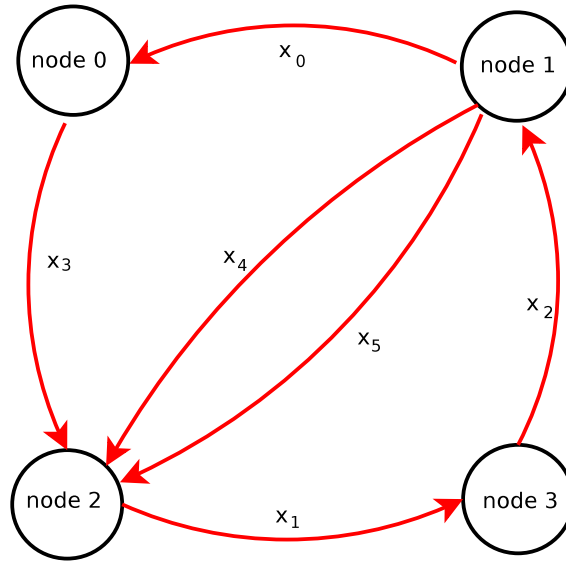


Figure 6.1: Simple network.

6.1.1.1 Source code

In this section we will show how to solve (6.2) with the network optimizer.

The Python program included below, which solves this problem, is distributed with MOSEK and can be found in the directory

```
mosek\6\tools\examples\python
```

```

1  #
2  # Copyright: Copyright (c) 1998-2011 MOSEK ApS, Denmark. All rights reserved.
3  #
4  # File:    network1.py
5  #
6  # Demonstrates a simple use of the network optimizer.
7  #
8  # Purpose: 1. Specify data for a network.
9  #           2. Solve the network problem with the network optimizer.
10 #
11
12 import sys
13 import mosek
14 from numpy import array, float, zeros
15
16 # Define a stream printer to grab output from MOSEK
17 def streamprinter(text):
18     sys.stdout.write(text)
19     sys.stdout.flush()
20
21 # Since the actual value of Infinity is ignored, we define it solely
22 # for symbolic purposes:
23 inf = 0.0
24
25 # Make mosek environment.
26 env = mosek.Env ()
27
28 # Initialize the environment.
29 env.init ()
30
31 # Attach a printer to the environment
32 env.set_Stream (mosek.streamtype.log, streamprinter)
33
34 numcon = 4
35 numvar = 6
36
37 # Specify network graph data.
38 netfrom = array ([0,2,3,1,1,1])
39 netto   = array ([2,3,1,0,2,2])
40
41 # Specify arc cost.
42 cc      = zeros (4, float)
43 cx      = array ([1.0,0.0,1.0,0.0,-1.0,1.0])
44
45 # Specify boundkeys.
46 bkc     = [mosek.boundkey.fx]*4
47
48 bkc     = [mosek.boundkey.lo]*6
49
50 # Specify bounds.
51 blc     = array ([1.0,1.0,-2.0,0.0])
52 buc     = array ([1.0,1.0,-2.0,0.0])
53 blx     = zeros (6, float)
54 bux     = array ([inf,inf,inf,inf,inf,inf])
55
56 # Specify zero primal solution.
57 xc      = zeros (4, float)

```

```

58 xx      = zeros (6, float)
59
60 # Specify zero dual solution.
61 y        = zeros (4, float)
62 slc      = zeros (4, float)
63 suc      = zeros (4, float)
64 slx      = zeros (6, float)
65 sux      = zeros (6, float)
66
67 # Specify status keys.
68 skc      = [mosek.stakey.unk]*4
69
70 skx      = [mosek.stakey.unk]*6
71
72 # Create a task object linked with the environment env.
73 dummytask = env.Task (numcon,numvar)
74
75 # Set the problem to be maximized
76 dummytask.putobjsense (mosek.objsense.maximize)
77
78 # Solve the network problem
79 prosta,solsta = dummytask.netoptimize(
80     cc,
81     cx,
82     bkc,
83     blc,
84     buc,
85     bkc,
86     blx,
87     bux,
88     netfrom,
89     netto,
90     0,
91     skc,
92     skx,
93     xc,
94     xx,
95     y,
96     slc,
97     suc,
98     slx,
99     sux)
100
101 if solsta == mosek.solsta.optimal :
102     print "Network problem is optimal"
103
104     print "Primal solution is :"
105     for i in range(0,numcon) :
106         print "xc[%d] = %-16.10e" % (i,xc[i])
107
108     for j in range(0,numvar) :
109         print "Arc(%d,%d) -> xx[%d] = %-16.10e" % (netfrom[j],netto[j],j,xx[j])
110 elif solsta == mosek.solsta.prim_infeas_cer :
111     print "Network problem is primal infeasible"
112 elif solsta == mosek.solsta.dual_infeas_cer :
113     print "Network problem is dual infeasible"
114 else :
115     print "Network problem solsta : %s" % solsta

```

6.1.1.2 Example code comments

There are a few important differences between the linear network optimization example in section 6.1.1.1 and the general linear optimization problem in section 5.2.

- MOSEK allows that network problems can be inputted and optimized using one function call to the function `Task.netoptimize`. This is more efficient and uses less memory than a call to the standard optimizer.
- Since we know that each column of matrix A has two non-zeroes, it can be stored in two arrays, `from` and `to`, specifying the origin and destination of the arcs (variables), see graph in fig.fig-network.
- The solution is written directly to `skc`, `skx`, `xc`, `xx`, `y`, `slc`, `suc`, `slx` and `sux` by `Task.netoptimize`.

6.2 Embedded network flow problems

Often problems contains both large parts with network structure and some non-network constraints or variables — such problems are said to have *embedded network structure*.

A linear optimization with embedded network structure problem can be written as :

$$\begin{array}{llll} \text{minimize} & & c^T x + c^f & \\ \text{subject to} & l_N^c \leq Nx \leq u_N^c, & & \\ & l^c \leq Ax \leq u^c, & & \\ & l^x \leq x \leq u^x, & & \end{array} \quad (6.3)$$

Where the constraints

$$l_N^c \leq Nx \leq u_N^c \quad (6.4)$$

defines a network as explained in section 6.1, and the constraints

$$l^c \leq Ax \leq u^c \quad (6.5)$$

defines the general non-network linear constraints. As an example consider the small linear optimization problem

$$\begin{array}{llllllllll} \text{maximize} & -x_0 & & + & x_2 & & - & x_4 & + & x_5 & & \\ \text{subject to} & 0.50x_0 & & & & + & 0.50x_3 & & & & = & 0.5, \\ & & & & 0.50x_2 & - & 0.50x_3 & + & 0.50x_4 & + & 0.50x_5 & = -1, \\ & -0.25x_0 & + & -2.50x_1 & + & & & - & 0.25x_4 & - & 0.25x_5 & = 0, \\ & & & 2.50x_1 & - & 0.25x_2 & & & & & & = 0, \\ & & - & x_1 & + & x_2 & + & x_3 & & + & x_5 & \geq 6, \end{array} \quad (6.6)$$

with the bounds

$$-\infty \leq x_0 \leq 0, 0 \leq x_j \leq \infty \text{ for } j = 1 \dots 5.$$

Recalling the network flow problem structural properties from section 6.1, each variable should appear in exactly two constraints with coefficients of either -1.0 or $+1.0$.

At first glance it does not seem to contain any network structure, but if we scale constraints 1-4 by respectively 2.0, 2.0, 4.0, 4.0 and columns 1-2 by -1.0, 0.1 we get the following problem :

$$\begin{array}{rcll} \text{maximize} & x_0 & + & x_2 & + & & - & x_4 & + & x_5 & & & \\ \text{subject to} & -x_0 & & & & + & x_3 & & & & & & = & 1, \\ & & & x_2 & - & x_3 & + & x_4 & + & x_5 & & & = & -2, \\ & x_0 & - & x_1 & & & & - & x_4 & - & x_5 & & = & 0, \\ & & x_1 & - & x_2 & + & & & & & & & = & 0, \\ & & x_1 & + & x_2 & + & x_3 & & & + & x_5 & & \geq & 6, \end{array} \quad (6.7)$$

with the bounds

$$0 \leq x_j \leq \infty \text{ for } j = 0 \dots 5.$$

This corresponds to the network flow problem in section 6.1.1 plus one extra non-network constraint. We cannot use the network optimizer directly on the above problem since the last constraint destroys the network property. Finding the largest possible network structure in a linear optimization problem is computationally difficult, so MOSEK offers a heuristic **Task.netextraction** that attempts to find suitable scaling factors maximizing numbers of network constraints and variables. Assuming that the embedded network structure is dominant and the problem has few non-network constraints, we can exploit this structure and potentially speed up the optimization. Since the network constraints can be handled efficiently by the specialized network optimizer, the following idea is used:

- Disregard the non-network constraints and optimize the network problem.
- Use the network solution to hot-start the standard dual simplex optimizer.

An embedded network can be exploited by this scheme in two ways:

- Use the extraction heuristics directly by the **Task.netextraction** function and optimize with the **Task.netoptimize** function.
- Let the simplex optimizer exploit embedded network structure automatically.

The first method is more difficult than the second, but also offers much more flexibility. In 6.2.1 the first method is demonstrated by a code example below. For further details on exploiting embedded network structure in the standard simplex optimizer, see section 8.3.1.

6.2.1 Example: Exploit embedded network flow structure in the simplex optimizer

MOSEK is distributed with some network examples which can be found in the directory

mosek\6\tools\examples

The example given in this section demonstrates how to extract and optimize embedded network structure in a arbitrary linear optimization problem. The following idea is used

- Read an arbitrary linear optimization problem into a task.
- Use the `Task.netextraction` function to extract embedded network structure.
- Optimize the network problem using the `Task.netoptimize` function.

```

1  #
2  # Copyright: Copyright (c) 1998-2011 MOSEK ApS, Denmark. All rights reserved.
3  #
4  # File:      network2.py
5  #
6  # Demonstrates a simple use of network structure in a model.
7  #
8  # Purpose: 1. Read an optimization problem from an
9  #           user specified MPS file.
10 #           2. Extract the embedded network (if any ).
11 #           3. Solve the embedded network with the network optimizer.
12 #
13 # Note that the general simplex optimizer called though MSK_optimize can also extract
14 # embedded network and solve it with the network optimizer. The direct call to the
15 # network optimizer, which is demonstrated here, is offered as an option to save
16 # memory and overhead for solving either many or large network problems.
17 #
18
19 import sys
20 import mosek
21 from numpy import resize,array, float, zeros
22 #from mosek.array import *
23
24 # Define a stream printer to grab output from MOSEK
25 def streamprinter(text):
26     sys.stdout.write(text)
27     sys.stdout.flush()
28
29 # Since the actual value of Infinity is ignores, we define it solely
30 # for symbolic purposes:
31 inf = 0.0
32
33 if len(sys.argv) != 2:
34     print "Wrong arguments. The syntax is:"
35     print " network2 inputfile"
36 else:
37     # Make mosek environment.
38     env = mosek.Env ()
39
40     # Initialize the environment.
41     env.init ()
42
43     # Attach a printer to the environment
44     env.set_Stream (mosek.streamtype.log, streamprinter)
45

```

```

46 # since we don't know the size of the problem.
47 numcon = []
48 numvar = []
49 netnumcon = []
50 netnumvar = []
51 task = env.Task (0,0)
52
53 task.readdata (sys.argv[1])
54
55 numcon = task.getnumcon ()
56 numvar = task.getnumvar ()
57
58 # Specify network graph data.
59 netfrom = zeros (numvar, int)
60 netto = zeros (numvar, int)
61
62 # Specify arc cost.
63 cc = zeros (numcon, float)
64 cx = zeros (numvar, float)
65
66 # Specify boundkeys.
67 bkc = [ mosek.boundkey.fx ] * numcon
68 bkx = [ mosek.boundkey.fx ] * numvar)
69
70 # Specify bounds.
71 blc = zeros (numcon, float)
72 buc = zeros (numcon, float)
73 blx = zeros (numvar, float)
74 bux = zeros (numvar, float)
75
76 # Specify data for extracted network.
77 scalcon = zeros (numcon, float)
78 scalvar = zeros (numvar, float)
79 netcon = zeros (numcon, int)
80 netvar = zeros (numvar, int)
81
82 # Extract embedded network
83 netnumcon,netnumvar = task.netextraction(
84     netcon,
85     netvar,
86     scalcon,
87     scalvar,
88     cx,
89     bkc,
90     blc,
91     buc,
92     bkx,
93     blx,
94     bux,
95     netfrom,
96     netto)
97
98 # Create a dummy task object linked with the environment env.
99 dummytask = env.Task (netnumcon,netnumvar)
100
101 # Array length for netoptimize must match netnumcon and netnumvar
102
103 # Resize network graph data.

```

```

104     netfrom = resize (netfrom,netnumvar)
105     netto   = resize (netto,netnumvar)
106
107     # Resize arc cost.
108     cc      = resize (cc,netnumcon)
109     cx      = resize (cx,netnumvar)
110
111     # Resize boundkeys.
112     bkc     = [ mosek.boundkey.fx ] * netnumcon
113     bkx     = [ mosek.boundkey.fx ] * netnumvar
114
115     # Resize bounds.
116     blc     = resize (blc,netnumcon)
117     buc     = resize (buc,netnumcon)
118     blx     = resize (blx,netnumvar)
119     bux     = resize (bux,netnumvar)
120
121     # Specify zero primal solution.
122     xc      = zeros (netnumcon, float)
123     xx      = zeros (netnumvar, float)
124
125     # Specify zero dual solution.
126     y       = zeros (netnumcon, float)
127     slc     = zeros (netnumcon, float)
128     suc     = zeros (netnumcon, float)
129     slx     = zeros (netnumvar, float)
130     sux     = zeros (netnumvar, float)
131
132     # Specify status keys.
133     skc     = [ mosek.stakey.unk ] * netnumcon
134     skx     = [ mosek.stakey.unk ] * netnumvar
135
136     # Specify problem and solution status.
137     prosta  = []
138     solsta  = []
139
140     # Solve the network problem
141     prosta,solsta = dummytask.netoptimize(
142         cc,
143         cx,
144         bkc,
145         blc,
146         buc,
147         bkx,
148         blx,
149         bux,
150         netfrom,
151         netto,
152         0,
153         skc,
154         skx,
155         xc,
156         xx,
157         y,
158         slc,
159         suc,
160         slx,
161         sux)

```

```
162     print "Original problem size : numcon : %d numvar : %d" % (numcon,numvar)
163     print "Embedded network size : numcon : %d numvar : %d" % (netnumcon,netnumvar)
164
165     if solsta == mosek.solsta.optimal :
166         print "Network problem is optimal"
167     elif solsta == mosek.solsta.prim_infeas_cer :
168         print "Network problem is primal infeasible"
169     elif solsta == mosek.solsta.dual_infeas_cer :
170         print "Network problem is dual infeasible"
171     else :
172         print "Network problem solsta : %s" % solsta
173
```

In the above example we only optimize the embedded network problem. We still need to use the found network solution as a hot-start for the simplex optimizer and solve the original problem. This involves unscaling the network solution back to same unit measure as the original problem. In the example

```
mosek\6\tools\examples\python\network3.py
```

we show how to convert the network solution into a valid hot-start for the simplex optimizer.

Chapter 7

Modelling

In this chapter we will discuss the following issues:

- The formal definitions of the problem types that MOSEK can solve.
- The solution information produced by MOSEK.
- The information produced by MOSEK if the problem is infeasible.
- A set of examples showing different ways of formulating commonly occurring problems so that they can be solved by MOSEK.
- Recommendations for formulating optimization problems.

7.1 Linear optimization

A linear optimization problem can be written as

$$\begin{array}{ll} \text{minimize} & c^T x + c^f \\ \text{subject to} & \begin{array}{ll} l^c \leq Ax \leq u^c, \\ l^x \leq x \leq u^x, \end{array} \end{array} \quad (7.1)$$

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.
- $A \in \mathbb{R}^{m \times n}$ 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.

A primal solution (x) is *(primal) feasible* if it satisfies all constraints in (7.1). If (7.1) has at least one primal feasible solution, then (7.1) is said to be (primal) feasible.

In case (7.1) does not have a feasible solution, the problem is said to be *(primal) infeasible*.

7.1.1 Duality for linear optimization

Corresponding to the primal problem (7.1), there is a dual problem

$$\begin{aligned} & \text{maximize} && (l^c)^T s_l^c - (u^c)^T s_u^c \\ & && + (l^x)^T s_l^x - (u^x)^T s_u^x + c^f \\ & \text{subject to} && A^T y + s_l^c - s_u^c = c, \\ & && -y + s_l^x - s_u^x = 0, \\ & && s_l^c, s_u^c, s_l^x, s_u^x \geq 0. \end{aligned} \tag{7.2}$$

If a bound in the primal problem is plus or minus infinity, the corresponding dual variable is fixed at 0, and we use the convention that the product of the bound value and the corresponding dual variable is 0. E.g.

$$l_j^x = -\infty \Rightarrow (s_l^x)_j = 0 \text{ and } l_j^x \cdot (s_l^x)_j = 0.$$

This is equivalent to removing variable $(s_l^x)_j$ from the dual problem.

A solution

$$(y, s_l^c, s_u^c, s_l^x, s_u^x)$$

to the dual problem is feasible if it satisfies all the constraints in (7.2). If (7.2) has at least one feasible solution, then (7.2) is *(dual) feasible*, otherwise the problem is *(dual) infeasible*.

We will denote a solution

$$(x, y, s_l^c, s_u^c, s_l^x, s_u^x)$$

so that x is a solution to the primal problem (7.1), and

$$(y, s_l^c, s_u^c, s_l^x, s_u^x)$$

is a solution to the corresponding dual problem (7.2). A solution which is both primal and dual feasible is denoted a *primal-dual feasible solution*.

7.1.1.1 A primal-dual feasible solution

Let

$$(x^*, y^*, (s_l^c)^*, (s_u^c)^*, (s_l^x)^*, (s_u^x)^*)$$

be a primal-dual feasible solution, and let

$$(x^c)^* := Ax^*.$$

For a primal-dual feasible solution we define the *optimality gap* as the difference between the primal and the dual objective value,

$$\begin{aligned} & c^T x^* + c^f - ((l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x + c^f) \\ &= \sum_{i=1}^m ((s_l^c)_i^* ((x_i^c)^* - l_i^c) + (s_u^c)_i^* (u_i^c - (x_i^c)^*)) + \sum_{j=1}^n ((s_l^x)_j^* (x_j - l_j^x) + (s_u^x)_j^* (u_j^x - x_j^*)) \\ &\geq 0 \end{aligned}$$

where the first relation can be obtained by multiplying the dual constraints (7.2) by x and x^c respectively, and the second relation comes from the fact that each term in each sum is nonnegative. It follows that the primal objective will always be greater than or equal to the dual objective.

We then define the *duality gap* as the difference between the primal objective value and the dual objective value, i.e.

$$c^T x^* + c^f - ((l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x + c^f)$$

Please note that the duality gap will always be nonnegative.

7.1.1.2 An optimal solution

It is well-known that a linear optimization problem has an optimal solution if and only if there exist feasible primal and dual solutions so that the duality gap is zero, or, equivalently, that the *complementarity conditions*

$$\begin{aligned} (s_l^c)_i^* ((x_i^c)^* - l_i^c) &= 0, & i = 1, \dots, m, \\ (s_u^c)_i^* (u_i^c - (x_i^c)^*) &= 0, & i = 1, \dots, m, \\ (s_l^x)_j^* (x_j - l_j^x) &= 0, & j = 1, \dots, n, \\ (s_u^x)_j^* (u_j^x - x_j^*) &= 0, & j = 1, \dots, n \end{aligned}$$

are satisfied.

If (7.1) has an optimal solution and MOSEK solves the problem successfully, both the primal and dual solution are reported, including a status indicating the exact state of the solution.

7.1.1.3 Primal infeasible problems

If the problem (7.1) is infeasible (has no feasible solution), MOSEK will report a certificate of primal infeasibility: The dual solution reported is a certificate of infeasibility, and the primal solution is undefined.

A certificate of primal infeasibility is a feasible solution to the modified dual problem

$$\begin{aligned} & \text{maximize} && (l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x \\ & \text{subject to} && A^T y + s_l^x - s_u^x = 0, \\ & && -y + s_l^c - s_u^c = 0, \\ & && s_l^c, s_u^c, s_l^x, s_u^x \geq 0. \end{aligned} \tag{7.3}$$

so that the objective is strictly positive, i.e. a solution

$$(y^*, (s_l^c)^*, (s_u^c)^*, (s_l^x)^*, (s_u^x)^*)$$

to (7.3) so that

$$(l^c)^T (s_l^c)^* - (u^c)^T (s_u^c)^* + (l^x)^T (s_l^x)^* - (u^x)^T (s_u^x)^* > 0.$$

Such a solution implies that (7.3) is unbounded, and that its dual is infeasible.

We note that the dual of (7.3) is a problem which constraints are identical to the constraints of the original primal problem (7.1): If the dual of (7.3) is infeasible, so is the original primal problem.

7.1.1.4 Dual infeasible problems

If the problem (7.2) is infeasible (has no feasible solution), MOSEK will report a certificate of dual infeasibility: The primal solution reported is a certificate of infeasibility, and the dual solution is undefined.

A certificate of dual infeasibility is a feasible solution to the problem

$$\begin{aligned} & \text{minimize} && c^T x \\ & \text{subject to} && Ax - x^c = 0, \\ & && \bar{l}^c \leq x^c \leq \bar{u}^c, \\ & && \bar{l}^x \leq x \leq \bar{u}^x \end{aligned} \tag{7.4}$$

where

$$\bar{l}_i^c = \begin{cases} 0, & \text{if } l_i^c > -\infty, \\ -\infty & \text{otherwise} \end{cases} \quad \text{and} \quad \bar{u}_i^c := \begin{cases} 0, & \text{if } u_i^c < \infty, \\ \infty & \text{otherwise} \end{cases}$$

and

$$\bar{l}_j^x = \begin{cases} 0, & \text{if } l_j^x > -\infty, \\ -\infty & \text{otherwise} \end{cases} \quad \text{and} \quad \bar{u}_j^x := \begin{cases} 0, & \text{if } u_j^x < \infty, \\ \infty & \text{otherwise} \end{cases}$$

so that the objective value $c^T x$ is negative. Such a solution implies that (7.4) is unbounded, and that the dual of (7.4) is infeasible.

We note that the dual of (7.4) is a problem which constraints are identical to the constraints of the original dual problem (7.2): If the dual of (7.4) is infeasible, so is the original dual problem.

7.1.2 Primal and dual infeasible case

In case that both the primal problem (7.1) and the dual problem (7.2) are infeasible, MOSEK will report only one of the two possible certificates — which one is not defined (MOSEK returns the first certificate found).

7.2 Quadratic and quadratically constrained optimization

A convex quadratic optimization problem is an optimization problem of the form

$$\begin{aligned} & \text{minimize} && \frac{1}{2}x^T Q^o x + c^T x + c^f \\ & \text{subject to} && l_k^c \leq \frac{1}{2}x^T Q^k x + \sum_{j=0}^{n-1} a_{k,i} x_j \leq u_k^c, \quad k = 0, \dots, m-1, \\ & && l^x \leq x \leq u^x, \quad j = 0, \dots, n-1, \end{aligned} \quad (7.5)$$

where the convexity requirement implies that

- Q^o is a symmetric positive semi-definite matrix.
- If $l_k^c = -\infty$, then Q^k is a symmetric positive semi-definite matrix.
- If $u_k^c = \infty$, then Q^k is a symmetric negative semi-definite matrix.
- If $l_k > -\infty$ and $u_k^k < \infty$, then Q^k is a zero matrix.

The convexity requirement is very important and it is strongly recommended that MOSEK is applied to convex problems only.

7.2.1 A general recommendation

Any convex quadratic optimization problem can be reformulated as a conic optimization problem. It is our experience that for the majority of practical applications it is better to cast them as conic problems because

- the resulting problem is convex by construction, and
- the conic optimizer is more efficient than the optimizer for general quadratic problems.

See Section 7.3.3.1 for further details.

7.2.2 Reformulating as a separable quadratic problem

The simplest quadratic optimization problem is

$$\begin{aligned} & \text{minimize} && 1/2x^T Qx + c^T x \\ & \text{subject to} && Ax = b, \\ & && x \geq 0. \end{aligned} \quad (7.6)$$

The problem (7.6) is said to be a separable problem if Q is a diagonal matrix or, in other words, if the quadratic terms in the objective all have this form

$$x_j^2$$

instead of this form

$$x_j x_i.$$

The separable form has the following advantages:

- It is very easy to check the convexity assumption, and
- the simpler structure in a separable problem usually makes it easier to solve.

It is well-known that a positive semi-definite matrix Q can always be factorized, i.e. a matrix F exists so that

$$Q = F^T F. \quad (7.7)$$

In many practical applications of quadratic optimization F is known explicitly; e.g. if Q is a covariance matrix, F is the set of observations producing it.

Using (7.7), the problem (7.6) can be reformulated as

$$\begin{aligned} & \text{minimize} && 1/2 y^T I y + c^T x \\ & \text{subject to} && \begin{aligned} A x &= b, \\ F x - y &= 0, \\ x &\geq 0. \end{aligned} \end{aligned} \quad (7.8)$$

The problem (7.8) is also a quadratic optimization problem and has more constraints and variables than (7.6). However, the problem is separable. Normally, if F has fewer rows than columns, it is worthwhile to reformulate as a separable problem. Indeed consider the extreme case where F has one dense row and hence Q will be a dense matrix.

The idea presented above is applicable to quadratic constraints too. Now, consider the constraint

$$1/2 x^T (F^T F) x \leq b \quad (7.9)$$

where F is a matrix and b is a scalar. (7.9) can be reformulated as

$$\begin{aligned} 1/2 y^T I y &\leq b, \\ F x - y &= 0. \end{aligned}$$

It should be obvious how to generalize this idea to make any convex quadratic problem separable.

Next, consider the constraint

$$1/2 x^T (D + F^T F) x \leq b$$

where D is a positive semi-definite matrix, F is a matrix, and b is a scalar. We assume that D has a simple structure, e.g. that D is a diagonal or a block diagonal matrix. If this is the case, it may be worthwhile performing the reformulation

$$\begin{aligned} 1/2 ((x^T D x) + y^T I y) &\leq b, \\ F x - y &= 0. \end{aligned}$$

Now, the question may arise: When should a quadratic problem be reformulated to make it separable or near separable? The simplest rule of thumb is that it should be reformulated if the number of non-zeros used to represent the problem decreases when reformulating the problem.

7.3 Conic optimization

Conic optimization can be seen as a generalization of linear optimization. Indeed a conic optimization problem is a linear optimization problem plus a constraint of the form

$$x \in \mathcal{C}$$

where \mathcal{C} is a convex cone. A complete conic problem has the form

$$\begin{aligned} & \text{minimize} && c^T x + c^f \\ & \text{subject to} && \begin{array}{lll} l^c & \leq & Ax & \leq & u^c, \\ l^x & \leq & x & \leq & u^x, \\ & & x \in \mathcal{C}. \end{array} \end{aligned} \tag{7.10}$$

The cone \mathcal{C} can be a Cartesian product of p convex cones, i.e.

$$\mathcal{C} = \mathcal{C}_1 \times \cdots \times \mathcal{C}_p$$

in which case $x \in \mathcal{C}$ can be written as

$$x = (x_1, \dots, x_p), \quad x_1 \in \mathcal{C}_1, \dots, x_p \in \mathcal{C}_p$$

where each $x_t \in \mathbb{R}^{n_t}$. Please note that the n -dimensional Euclidean space \mathbb{R}^n is a cone itself, so simple linear variables are still allowed.

MOSEK supports only a limited number of cones, specifically

$$\mathcal{C} = \mathcal{C}_1 \times \cdots \times \mathcal{C}_p$$

where each \mathcal{C}_t has one of the following forms

- \mathbb{R} set:

$$\mathcal{C}_t = \{x \in \mathbb{R}^{n_t}\}.$$

- Quadratic cone:

$$\mathcal{C}_t = \left\{ x \in \mathbb{R}^{n_t} : x_1 \geq \sqrt{\sum_{j=2}^{n_t} x_j^2} \right\}.$$

- Rotated quadratic cone:

$$\mathcal{C}_t = \left\{ x \in \mathbb{R}^{n_t} : 2x_1x_2 \geq \sum_{j=3}^{n_t} x_j^2, \quad x_1, x_2 \geq 0 \right\}.$$

Although these cones may seem to provide only limited expressive power they can be used to model a large range of problems as demonstrated in Section [7.3.3](#).

7.3.1 Duality for conic optimization

The dual problem corresponding to the conic optimization problem (7.10) is given by

$$\begin{aligned}
& \text{maximize} && (l^c)^T s_l^c - (u^c)^T s_u^c \\
& && + (l^x)^T s_l^x - (u^x)^T s_u^x + c^f \\
& \text{subject to} && A^T y + s_l^x - s_u^x + s_n^x = c, \\
& && -y + s_l^c - s_u^c = 0, \\
& && s_l^c, s_u^c, s_l^x, s_u^x \geq 0, \\
& && s_n^x \in \mathcal{C}^*
\end{aligned} \tag{7.11}$$

where the dual cone \mathcal{C}^* is a product of the cones

$$\mathcal{C}^* = \mathcal{C}_1^* \times \dots \times \mathcal{C}_p^*$$

where each \mathcal{C}_t^* is the dual cone of \mathcal{C}_t . For the cone types MOSEK can handle, the relation between the primal and dual cone is given as follows:

- \mathbb{R} set:

$$\mathcal{C}_t = \{x \in \mathbb{R}^{n^t}\} \Leftrightarrow \mathcal{C}_t^* := \{s \in \mathbb{R}^{n^t} : s = 0\}.$$

- Quadratic cone:

$$\mathcal{C}_t := \left\{ x \in \mathbb{R}^{n^t} : x_1 \geq \sqrt{\sum_{j=2}^{n^t} x_j^2} \right\} \Leftrightarrow \mathcal{C}_t^* = \mathcal{C}_t.$$

- Rotated quadratic cone:

$$\mathcal{C}_t := \left\{ x \in \mathbb{R}^{n^t} : 2x_1x_2 \geq \sum_{j=3}^{n^t} x_j^2, x_1, x_2 \geq 0 \right\} \Leftrightarrow \mathcal{C}_t^* = \mathcal{C}_t.$$

Please note that the dual problem of the dual problem is identical to the original primal problem.

7.3.2 Infeasibility

In case MOSEK finds a problem to be infeasible it reports a certificate of the infeasibility. This works exactly as for linear problems (see Sections 7.1.1.3 and 7.1.1.4).

7.3.3 Examples

This section contains several examples of inequalities and problems that can be cast as conic optimization problems.

7.3.3.1 Quadratic objective and constraints

From Section 7.2.2 we know that any convex quadratic problem can be stated on the form

$$\begin{aligned} & \text{minimize} && 0.5 \|Fx\|^2 + c^T x, \\ & \text{subject to} && 0.5 \|Gx\|^2 + a^T x \leq b, \end{aligned} \tag{7.12}$$

where F and G are matrices and c and a are vectors. For simplicity we assume that there is only one constraint, but it should be obvious how to generalize the methods to an arbitrary number of constraints.

Problem (7.12) can be reformulated as

$$\begin{aligned} & \text{minimize} && 0.5 \|t\|^2 + c^T x, \\ & \text{subject to} && 0.5 \|z\|^2 + a^T x \leq b, \\ & && Fx - t = 0, \\ & && Gx - z = 0 \end{aligned} \tag{7.13}$$

after the introduction of the new variables t and z . It is easy to convert this problem to a conic quadratic optimization problem, i.e.

$$\begin{aligned} & \text{minimize} && v + c^T x, \\ & \text{subject to} && p + a^T x = b, \\ & && Fx - t = 0, \\ & && Gx - z = 0, \\ & && w = 1, \\ & && q = 1, \\ & && \|t\|^2 \leq 2vw, \quad v, w \geq 0, \\ & && \|z\|^2 \leq 2pq, \quad p, q \geq 0. \end{aligned} \tag{7.14}$$

In this case we can model the last two inequalities using rotated quadratic cones.

If we assume that F is a non-singular matrix — e.g. a diagonal matrix — then

$$x = F^{-1}t$$

and hence we can eliminate x from the problem to obtain:

$$\begin{aligned} & \text{minimize} && v + c^T F^{-1}t, \\ & \text{subject to} && p + a^T F^{-1}t = b, \\ & && GF^{-1}t - z = 0, \\ & && w = 1, \\ & && q = 1, \\ & && \|t\|^2 \leq 2vw, \quad v, w \geq 0, \\ & && \|z\|^2 \leq 2pq, \quad p, q \geq 0. \end{aligned} \tag{7.15}$$

In most cases MOSEK performs this reduction automatically during the presolve phase before the optimization is performed.

7.3.3.2 Minimizing a sum of norms

The next example is the problem of minimizing a sum of norms, i.e. the problem

$$\begin{aligned} & \text{minimize} && \sum_{i=1}^k \|x^i\| \\ & \text{subject to} && Ax = b, \end{aligned} \tag{7.16}$$

where

$$x := \begin{bmatrix} x^1 \\ \vdots \\ x^k \end{bmatrix}.$$

This problem is equivalent to

$$\begin{aligned} & \text{minimize} && \sum_{i=1}^k z_i \\ & \text{subject to} && Ax = b, \\ & && \|x^i\| \leq z_i, \quad i = 1, \dots, k, \end{aligned} \tag{7.17}$$

which in turn is equivalent to

$$\begin{aligned} & \text{minimize} && \sum_{i=1}^k z_i \\ & \text{subject to} && Ax = b, \\ & && (z_i, x^i) \in \mathcal{C}_i, \quad i = 1, \dots, k \end{aligned} \tag{7.18}$$

where all \mathcal{C}_i are of the quadratic type, i.e.

$$\mathcal{C}_i := \{(z_i, x^i) : z_i \geq \|x^i\|\}.$$

The dual problem corresponding to (7.18) is

$$\begin{aligned} & \text{maximize} && b^T y \\ & \text{subject to} && A^T y + s = c, \\ & && t_i = 1, \quad i = 1, \dots, k, \\ & && (t_i, s^i) \in \mathcal{C}_i, \quad i = 1, \dots, k \end{aligned} \tag{7.19}$$

where

$$s := \begin{bmatrix} s^1 \\ \vdots \\ s^k \end{bmatrix}.$$

This problem is equivalent to

$$\begin{aligned} & \text{maximize} && b^T y \\ & \text{subject to} && A^T y + s = c, \\ & && \|s^i\|_2^2 \leq 1, \quad i = 1, \dots, k. \end{aligned} \tag{7.20}$$

Please note that in this case the dual problem can be reduced to an “ordinary” convex quadratically constrained optimization problem due to the special structure of the primal problem. In some cases it turns out that it is much better to solve the dual problem (7.19) rather than the primal problem (7.18).

7.3.3.3 Modelling polynomial terms using conic optimization

Generally an arbitrary polynomial term of the form

$$fx^g$$

cannot be represented with conic quadratic constraints, however in the following we will demonstrate some special cases where it is possible.

A particular simple polynomial term is the reciprocal, i.e.

$$\frac{1}{x}.$$

Now, a constraint of the form

$$\frac{1}{x} \leq y$$

where it is required that $x > 0$ is equivalent to

$$1 \leq xy \text{ and } x > 0$$

which in turn is equivalent to

$$\begin{aligned} z &= \sqrt{2}, \\ z^2 &\leq 2xy. \end{aligned}$$

The last formulation is a conic constraint plus a simple linear equality.

E.g., consider the problem

$$\begin{aligned} &\text{minimize} && c^T x \\ &\text{subject to} && \sum_{j=1}^n \frac{f_j}{x_j} \leq b, \\ &&& x \geq 0, \end{aligned}$$

where it is assumed that $f_j > 0$ and $b > 0$. This problem is equivalent to

$$\begin{aligned} &\text{minimize} && c^T x \\ &\text{subject to} && \sum_{j=1}^n f_j z_j = b, \\ &&& v_j = \sqrt{2}, \quad j = 1, \dots, n, \\ &&& v_j^2 \leq 2z_j x_j, \quad j = 1, \dots, n, \\ &&& x, z \geq 0, \end{aligned} \tag{7.21}$$

because

$$v_j^2 = 2 \leq 2z_j x_j$$

implies that

$$\frac{1}{x_j} \leq z_j \text{ and } \sum_{j=1}^n \frac{f_j}{x_j} \leq \sum_{j=1}^n f_j z_j = b.$$

The problem (7.21) is a conic quadratic optimization problem having n 3-dimensional rotated quadratic cones.

The next example is the constraint

$$\begin{aligned}\sqrt{x} &\geq |t|, \\ x &\geq 0,\end{aligned}$$

where both t and x are variables. This set is identical to the set

$$\begin{aligned}t^2 &\leq 2xz, \\ z &= 0.5, \\ x, z, &\geq 0.\end{aligned}\tag{7.22}$$

Occasionally, when modeling the *market impact* term in portfolio optimization, the polynomial term $x^{\frac{3}{2}}$ occurs. Therefore, consider the set defined by the inequalities

$$\begin{aligned}x^{1.5} &\leq t, \\ 0 &\leq x.\end{aligned}\tag{7.23}$$

We will exploit that $x^{1.5} = x^2/\sqrt{x}$. First define the set

$$\begin{aligned}x^2 &\leq 2st, \\ s, t &\geq 0.\end{aligned}\tag{7.24}$$

Now, if we can make sure that

$$2s \leq \sqrt{x},$$

then we have the desired result since this implies that

$$x^{1.5} = \frac{x^2}{\sqrt{x}} \leq \frac{x^2}{2s} \leq t.$$

Please note that s can be chosen freely and that $\sqrt{x} = 2s$ is a valid choice.

Let

$$\begin{aligned}x^2 &\leq 2st, \\ w^2 &\leq 2vr, \\ x &= v, \\ s &= w, \\ r &= \frac{1}{8}, \\ s, t, v, r &\geq 0,\end{aligned}\tag{7.25}$$

then

$$\begin{aligned}s^2 &= w^2 \\ &\leq 2vr \\ &= \frac{v}{4} \\ &= \frac{x}{4}.\end{aligned}$$

Moreover,

$$\begin{aligned}x^2 &\leq 2st, \\ &\leq 2\sqrt{\frac{x}{4}}t\end{aligned}$$

leading to the conclusion that

$$x^{1.5} \leq t.$$

(7.25) is a conic reformulation which is equivalent to (7.23). Please note that the $x \geq 0$ constraint does not appear explicitly in (7.24) and (7.25), but implicitly since $x = v \geq 0$.

As we shall see next, any polynomial term of the form x^g where g is a positive rational number can be represented using conic quadratic constraints [3, pp. 12-13], [12].

7.3.3.4 Optimization with rational polynomials

We next demonstrate how to model convex polynomial constraints of the form $x^{p/q} \leq t$ (where p and q are both positive integers) as a set of rotated quadratic cone constraints.

Following Ben-Tal et al. [12, p. 105] we use an intermediate result, namely that the set

$$\{s \in \mathbb{R}, y \in \mathbb{R}_+^{2^l} \mid s \leq (2^{l2^{l-1}} y_1 y_2 \cdots y_{2^l})^{1/2^l}\}$$

is convex and can be represented as a set of rotated quadratic cone constraints. To see this, we rewrite the condition (exemplified for $l = 3$),

$$s \leq (2^{12} \cdot y_1 \cdot y_2 \cdot y_3 \cdot y_4 \cdot y_5 \cdot y_6 \cdot y_7 \cdot y_8)^{1/8} \quad (7.26)$$

as

$$s^8 \leq (2^{12} \cdot y_1 \cdot y_2 \cdot y_3 \cdot y_4 \cdot y_5 \cdot y_6 \cdot y_7 \cdot y_8) \quad (7.27)$$

since all $y_i \geq 0$. We next introduce l levels of auxiliary variables and (rotated cone) constraints

$$y_{11}^2 \leq 2y_1 y_2, \quad y_{12}^2 \leq 2y_3 y_4, \quad y_{13}^2 \leq 2y_5 y_6, \quad y_{14}^2 \leq 2y_7 y_8, \quad (7.28)$$

$$y_{21}^2 \leq 2y_{11} y_{12}, \quad y_{22}^2 \leq 2y_{13} y_{14}, \quad (7.29)$$

and finally

$$s^2 \leq 2y_{21} y_{22}. \quad (7.30)$$

By simple substitution we see that (7.30) and (7.27) are equivalent, and since (7.30) involves only a set of simple rotated conic constraints then the original constraint (7.26) can be represented using only rotated conic constraints.

7.3.3.5 Convex increasing power functions

Using the intermediate result in section 7.3.3.4 we can include convex power functions with positive rational powers, i.e., constraints of the form

$$x^{p/q} \leq t, \quad x \geq 0$$

where p and q are positive integers and $p/q \geq 1$. For example, consider the constraints

$$x^{5/3} \leq t, \quad x \geq 0.$$

We rewrite it as

$$x^8 \leq x^3 t^3, \quad x \geq 0$$

which in turn is equivalent to

$$x^8 \leq 2^{12} y_1 y_2 \cdots y_8, \quad x = y_1 = y_2 = y_3, \quad y_4 = y_5 = y_6 = t, \quad y_7 = 1, \quad y_8 = 2^{-12}, \quad x, y_i \geq 0,$$

i.e., it can be represented as a set of rotated conic and linear constraints using the reformulation above.

For general p and q we choose l as the smallest integer such that $p \leq 2^l$ and we construct the problem as

$$x^{2^l} \leq 2^{l2^{l-1}} y_1 y_2 \cdots y_{2^l}, \quad x, y_i \geq 0,$$

with the first $2^l - p$ elements of y set to x , the next q elements set to t , and the product of the remaining elements as $1/2^{l2^{l-1}}$, i.e.,

$$x^{2^l} \leq x^{2^l - p} t^q, \quad x \geq 0 \quad \Longleftrightarrow \quad x^{p/q} \leq t, \quad x \geq 0.$$

7.3.3.6 Decreasing power functions

We can also include decreasing power functions with positive rational powers

$$x^{-p/q} \leq t, \quad x \geq 0$$

where p and q are positive integers. For example, consider

$$x^{-5/2} \leq t, \quad x \geq 0,$$

or equivalently

$$1 \leq x^5 t^2, \quad x \geq 0,$$

which, in turn, can be rewritten as

$$s^8 \leq 2^{12} y_1 y_2 \cdots y_8, \quad s = 2^{3/2}, \quad y_1 = \cdots = y_5 = x, \quad y_6 = y_7 = y_8 = t, \quad x, y_i \geq 0.$$

For general p and q we choose l as the smallest integer such that $p + q \leq 2^l$ and we construct the problem as

$$s^{2^l} \leq y_1 y_2 \cdots y_{2^l}, \quad y_i \geq 0,$$

with $s = 2^{l/2}$ and the first p elements of y set to x , the next q elements set to t , and the remaining elements set to 1, i.e.,

$$1 \leq x^p t^q, \quad x \geq 0 \quad \Longleftrightarrow \quad x^{-p/q} \leq t, \quad x \geq 0.$$

7.3.3.7 Minimizing general polynomials

Using the formulations in section 7.3.3.5 and section 7.3.3.6 it is straightforward to minimize general polynomials. For example, we can minimize

$$f(x) = x^2 + x^{-2}$$

which is used in statistical matching. We first formulate the problem

$$\begin{array}{ll} \text{minimize} & u + v \\ \text{subject to} & x^2 \leq u \\ & x^{-2} \leq v, \end{array}$$

which is equivalent to the quadratic conic optimization problem

$$\begin{aligned}
 & \text{minimize} && u + v \\
 & \text{subject to} && x^2 \leq 2uw \\
 & && s^2 \leq 2y_{21}y_{22} \\
 & && y_{21}^2 \leq 2y_1y_2 \\
 & && y_{22}^2 \leq 2y_3y_4 \\
 & && w = 1 \\
 & && s = 2^{3/4} \\
 & && y_1 = y_2 = x \\
 & && y_3 = v \\
 & && y_4 = 1
 \end{aligned}$$

in the variables $(x, u, v, w, s, y_1, y_2, y_3, y_4, y_{21}, y_{22})$.

7.3.3.8 Further reading

If you want to learn more about what can be modeled as a conic optimization problem we recommend the references [3, 12, 16].

7.3.4 Potential pitfalls in conic optimization

While a linear optimization problem either has a bounded optimal solution or is infeasible, the conic case is not as simple as that.

7.3.4.1 Non-attainment in the primal problem

Consider the example

$$\begin{aligned}
 & \text{minimize} && z \\
 & \text{subject to} && 2yz \geq x^2, \\
 & && x = \sqrt{2}, \\
 & && y, z \geq 0,
 \end{aligned} \tag{7.31}$$

which corresponds to the problem

$$\begin{aligned}
 & \text{minimize} && \frac{1}{y} \\
 & \text{subject to} && y \geq 0.
 \end{aligned} \tag{7.32}$$

Clearly, the optimal objective value is zero but it is never attained because implicitly we assume that the optimal y is finite.

7.3.4.2 Non-attainment in the dual problem

Next, consider the example

$$\begin{aligned}
 & \text{minimize} && x_4 \\
 & \text{subject to} && x_3 + x_4 = 1, \\
 & && x_1 = 0, \\
 & && x_2 = 1, \\
 & && 2x_1x_2 \geq x_3^2, \\
 & && x_1, x_2 \geq 0,
 \end{aligned} \tag{7.33}$$

which has the optimal solution

$$x_1^* = 0, x_2^* = 1, x_3^* = 0 \text{ and } x_4^* = 1$$

implying that the optimal primal objective value is 1.

Now, the dual problem corresponding to (7.33) is

$$\begin{aligned}
 & \text{maximize} && y_1 + y_3 \\
 & \text{subject to} && y_2 + s_1 = 0, \\
 & && y_3 + s_2 = 0, \\
 & && y_1 + s_3 = 0, \\
 & && y_1 = 1, \\
 & && 2s_1s_2 \geq s_3^2, \\
 & && s_1, s_2 \geq 0.
 \end{aligned} \tag{7.34}$$

Therefore,

$$y_1^* = 1$$

and

$$s_3^* = -1.$$

This implies that

$$2s_1^*s_2^* \geq (s_3^*)^2 = 1$$

and hence $s_2^* > 0$. Given this fact we can conclude that

$$\begin{aligned}
 y_1^* + y_3^* &= 1 - s_2^* \\
 &< 1
 \end{aligned}$$

implying that the optimal dual objective value is 1, however, this is never attained. Hence, no primal-dual bounded optimal solution with zero duality gap exists. Of course it is possible to find a primal-dual feasible solution such that the duality gap is close to zero, but then s_1^* will be similarly large. This is likely to make the problem (7.33) hard to solve.

An inspection of the problem (7.33) reveals the constraint $x_1 = 0$, which implies that $x_3 = 0$. If we either add the redundant constraint

$$x_3 = 0$$

to the problem (7.33) or eliminate x_1 and x_3 from the problem it becomes easy to solve.

7.4 Recommendations

Often an optimization problem can be formulated in several different ways, and the exact formulation used may have a significant impact on the solution time and the quality of the solution. In some cases the difference between a “good” and a “bad” formulation means the ability to solve the problem or not.

Below is a list of several issues that you should be aware of when developing a good formulation.

1. Sparsity is very important. The constraint matrix A is assumed to be a sparse matrix, where sparse means that it contains many zeros (typically less than 10% non-zeros). Normally, when A is sparser, less memory is required to store the problem and it can be solved faster.

2. Avoid large bounds as these can introduce all sorts of numerical problems. Assume that a variable x_j has the bounds

$$0.0 \leq x_j \leq 1.0e16.$$

The number 1.0e16 is large and it is very likely that the constraint $x_j \leq 1.0e16$ is non-binding at optimum, and therefore that the bound 1.0e16 will not cause problems. Unfortunately, this is a naïve assumption because the bound 1.0e16 may actually affect the presolve, the scaling, the computation of the dual objective value, etc. In this case the constraint $x_j \geq 0$ is likely to be sufficient, i.e. 1.0e16 is just a way of representing infinity.

3. Avoid large penalty terms in the objective, i.e. do not have large terms in the linear part of the objective function. They will most likely cause numerical problems.
4. On a computer all computations are performed in finite precision, which implies that

$$1 = 1 + \varepsilon$$

where ε is about 10^{-16} . This means that the results of all computations are truncated and therefore causing rounding errors. The upshot is that very small numbers and very large numbers should be avoided, e.g. it is recommended that all elements in A either are zero or belong to the interval $[10^{-6}, 10^6]$. The same holds for the bounds and the linear objective.

5. Decreasing the number of variables or constraints does not *necessarily* make it easier to solve a problem. In certain cases, i.e. in nonlinear optimization, it may be a good idea to introduce more constraints and variables if it makes the model separable. Furthermore, a big but sparse problem may be advantageous compared to a smaller but denser problem.
6. Try to avoid linearly dependent rows among the linear constraints. Network flow problems and multi-commodity network flow problems, for example, often contain one or more linearly dependent rows.
7. Finally, it is recommended to consult some of the papers about preprocessing to get some ideas about efficient formulations. See e.g. [4, 5, 14, 15].

7.4.1 Avoid near infeasible models

Consider the linear optimization problem

$$\begin{array}{ll} \text{minimize} & \\ \text{subject to} & \begin{array}{ll} x + y & \leq 10^{-10} + \alpha, \\ 1.0e4x + 2.0e4y & \geq 10^{-6}, \\ x, y & \geq 0. \end{array} \end{array} \quad (7.35)$$

Clearly, the problem is feasible for $\alpha = 0$. However, for $\alpha = -1.0e - 10$ the problem is infeasible. This implies that an insignificant change in the right side of the constraints makes the problem status switch from feasible to infeasible. Such a model should be avoided.

7.5 Examples continued

7.5.1 The absolute value

Assume that we have a constraint for the form

$$|f^T x + g| \leq b \quad (7.36)$$

where $x \in \mathbb{R}^n$ is a vector of variables, and $f \in \mathbb{R}^n$ and $g, b \in \mathbb{R}$ are constants.

It is easy to verify that the constraint (7.36) is equivalent to

$$-b \leq f^T x + g \leq b \quad (7.37)$$

which is a set of ordinary linear inequality constraints.

Please note that equalities involving an absolute value such as

$$|x| = 1$$

cannot be formulated as a linear or even a as convex nonlinear optimization problem. It requires integer constraints.

7.5.2 The Markowitz portfolio model

In this section we will show how to model several versions of the Markowitz portfolio model using conic optimization.

The Markowitz portfolio model deals with the problem of selecting a portfolio of assets, i.e. stocks, bonds, etc. The goal is to find a portfolio such that for a given return the risk is minimized. The assumptions are:

- A portfolio can consist of n traded assets numbered $1, 2, \dots$ held over a period of time.
- w_j^0 is the initial holding of asset j where $\sum_j w_j^0 > 0$.

- r_j is the return on asset j and is assumed to be a random variable. r has a known mean \bar{r} and covariance Σ .

The variable x_j denotes the amount of asset j traded in the given period of time and has the following meaning:

- If $x_j > 0$, then the amount of asset j is increased (by purchasing).
- If $x_j < 0$, then the amount of asset j is decreased (by selling).

The model deals with two central quantities:

- Expected return:

$$E[r^T(w^0 + x)] = \bar{r}^T(w^0 + x).$$

- Variance (Risk):

$$V[r^T(w^0 + x)] = (w^0 + x)^T \Sigma (w^0 + x).$$

By definition Σ is positive semi-definite and

$$\begin{aligned} \text{Std. dev.} &= \left\| \Sigma^{\frac{1}{2}}(w^0 + x) \right\| \\ &= \left\| L^T(w^0 + x) \right\| \end{aligned}$$

where L is **any** matrix such that

$$\Sigma = LL^T$$

A low rank of Σ is advantageous from a computational point of view. A valid L can always be computed as the Cholesky factorization of Σ .

7.5.2.1 Minimizing variance for a given return

In our first model we want to minimize the variance while selecting a portfolio with a specified expected target return t . Additionally, the portfolio must satisfy the budget (self-financing) constraint asserting that the total amount of assets sold must equal the total amount of assets purchased. This is expressed in the model

$$\begin{aligned} &\text{minimize} && V[r^T(w^0 + x)] \\ &\text{subject to} && E[r^T(w^0 + x)] = t, \\ &&& e^T x = 0, \end{aligned} \tag{7.38}$$

where $e := (1, \dots, 1)^T$. Using the definitions above this may be formulated as a quadratic optimization problem:

$$\begin{aligned} &\text{minimize} && (w^0 + x)^T \Sigma (w^0 + x) \\ &\text{subject to} && \bar{r}^T(w^0 + x) = t, \\ &&& e^T x = 0. \end{aligned} \tag{7.39}$$

7.5.2.2 Conic quadratic reformulation

An equivalent conic quadratic reformulation is given by:

$$\begin{aligned}
& \text{minimize} && f \\
& \text{subject to} && \Sigma^{\frac{1}{2}}(w^0 + x) - g = 0, \\
& && \bar{r}^T(w^0 + x) = t, \\
& && e^T x = 0, \\
& && f \geq \|g\|.
\end{aligned} \tag{7.40}$$

Here we minimize the standard deviation instead of the variance. Please note that $\Sigma^{\frac{1}{2}}$ can be replaced by any matrix L where $\Sigma = LL^T$. A low rank L is computationally advantageous.

7.5.2.3 Transaction costs with market impact term

We will now expand our model to include transaction costs as a fraction of the traded volume. [1, pp. 445-475] argues that transaction costs can be modeled as follows

$$\text{commission} + \frac{\text{bid}}{\text{ask}} - \text{spread} + \theta \sqrt{\frac{\text{trade volume}}{\text{daily volume}}}, \tag{7.41}$$

and that it is important to incorporate these into the model.

In the following we deal with the last of these terms denoted the *market impact term*. If you sell (buy) a lot of assets the price is likely to go down (up). This can be captured in the market impact term

$$\theta \sqrt{\frac{\text{trade volume}}{\text{daily volume}}} \approx m_j \sqrt{|x_j|}.$$

The θ and “daily volume” have to be estimated in some way, i.e.

$$m_j = \frac{\theta}{\sqrt{\text{daily volume}}}$$

has to be estimated. The market impact term gives the cost as a fraction of daily traded volume ($|x_j|$). Therefore, the total cost when trading an amount x_j of asset j is given by

$$|x_j|(m_j|x_j|^{\frac{1}{2}}).$$

This leads us to the model:

$$\begin{aligned}
& \text{minimize} && f \\
& \text{subject to} && \Sigma^{\frac{1}{2}}(w^0 + x) - g = 0, \\
& && \bar{r}^T(w^0 + x) = t, \\
& && e^T x + e^T y = 0, \\
& && |x_j|(m_j|x_j|^{\frac{1}{2}}) \leq y_j, \\
& && f \geq \|g\|.
\end{aligned} \tag{7.42}$$

Now, defining the variable transformation

$$y_j = m_j \bar{y}_j$$

we obtain

$$\begin{aligned} & \text{minimize} && f \\ & \text{subject to} && \Sigma^{\frac{1}{2}}(w^0 + x) - g = 0, \\ & && \bar{r}^T(w^0 + x) = t, \\ & && e^T x + m^T \bar{y} = 0, \\ & && |x_j|^{3/2} \leq \bar{y}_j, \\ & && f \geq \|g\|. \end{aligned} \tag{7.43}$$

As shown in Section 7.3.3.3 the set

$$|x_j|^{3/2} \leq \bar{y}_j$$

can be modeled by

$$\begin{aligned} x_j &\leq z_j, \\ -x_j &\leq z_j, \\ z_j^2 &\leq 2s_j \bar{y}_j, \\ u_j^2 &\leq 2v_j q_j, \\ z_j &= v_j, \\ s_j &= u_j, \\ q_j &= \frac{1}{8}, \\ q_j, s_j, \bar{y}_j, v_j, q_j &\geq 0. \end{aligned} \tag{7.44}$$

7.5.2.4 Further reading

For further reading please see [17] in particular, and [20] and [1], which also contain relevant material.

Chapter 8

The optimizers for continuous problems

The most essential part of MOSEK is the optimizers. Each optimizer is designed to solve a particular class of problems i.e. linear, conic, or general nonlinear problems. The purpose of the present chapter is to discuss which optimizers are available for the continuous problem classes and how the performance of an optimizer can be tuned, if needed.

This chapter deals with the optimizers for *continuous problems* with no integer variables.

8.1 How an optimizer works

When the optimizer is called, it roughly performs the following steps:

Presolve: Preprocessing to reduce the size of the problem.

Dualizer: Choosing whether to solve the primal or the dual form of the problem.

Scaling: Scaling the problem for better numerical stability.

Optimize: Solve the problem using selected method.

The first three preprocessing steps are transparent to the user, but useful to know about for tuning purposes. In general, the purpose of the preprocessing steps is to make the actual optimization more efficient and robust.

8.1.1 Presolve

Before an optimizer actually performs the optimization the problem is preprocessed using the so-called presolve. The purpose of the presolve is to

- remove redundant constraints,
- eliminate fixed variables,
- remove linear dependencies,
- substitute out free variables, and
- 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 [4, 5].

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 `mosek.iparam.presolve.use` to `mosek.presolvemode.off`.

The two most time-consuming steps of the presolve are

- the eliminator, and
- the linear dependency check.

Therefore, in some cases it is worthwhile to disable one or both of these.

8.1.1.1 Eliminator

The purpose of the eliminator is to eliminate free and implied free variables from the problem using substitution. For instance, given the constraints

$$\begin{aligned} y &= \sum_j x_j, \\ y, x &\geq 0, \end{aligned}$$

y is an implied free variable that can be substituted out of the problem, if deemed worthwhile.

If the eliminator consumes too much time or memory compared to the reduction in problem size gained it may be disabled. This can be done with the parameter `mosek.iparam.presolve_eliminator_use` to `mosek.onoffkey.off`.

8.1.1.2 Linear dependency checker

The purpose of the linear dependency check is to remove linear dependencies among the linear equalities. For instance, the three linear equalities

$$\begin{aligned} x_1 + x_2 + x_3 &= 1, \\ x_1 + 0.5x_2 &= 0.5, \\ 0.5x_2 + x_3 &= 0.5 \end{aligned}$$

contain exactly one linear dependency. This implies that one of the constraints can be dropped without changing the set of feasible solutions. Removing linear dependencies is in general a good idea since it reduces the size of the problem. Moreover, the linear dependencies are likely to introduce numerical problems in the optimization phase.

It is best practise to build models without linear dependencies. If the linear dependencies are removed at the modeling stage, the linear dependency check can safely be disabled by setting the parameter `mosek.iparam.presolve_lindep_use` to `mosek.onoffkey.off`.

8.1.2 Dualizer

All linear, conic, and convex optimization problems have an equivalent dual problem associated with them. MOSEK has built-in heuristics to determine if it is most efficient to solve the primal or dual problem. The form (primal or dual) solved is displayed in the MOSEK log. Should the internal heuristics not choose the most efficient form of the problem it may be worthwhile to set the dualizer manually by setting the parameters:

- `mosek.iparam.intpnt_solve_form`: In case of the interior-point optimizer.
- `mosek.iparam.sim_solve_form`: In case of the simplex optimizer.

Note that currently only linear problems may be dualized.

8.1.3 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 calculations. 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 to this problem is to reformulate it, 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

`mosek.iparam.intpnt_scaling` and `mosek.iparam.sim_scaling`

respectively.

8.1.4 Using multiple CPU's

The interior-point optimizers in MOSEK have been parallelized. This means that if you solve linear, quadratic, conic, or general convex optimization problem using the interior-point optimizer, you can take advantage of multiple CPU's.

By default MOSEK uses one thread to solve the problem, but the number of threads (and thereby CPUs) employed can be changed by setting the parameter `mosek.iparam.intpnt_num_threads`. This should never exceed the number of CPU's on the machine.

The speed-up obtained when using multiple CPUs is highly problem and hardware dependent, and consequently, it is advisable to compare single threaded and multi threaded performance for the given problem type to determine the optimal settings.

For small problems, using multiple threads will probably not be worthwhile.

8.2 Linear optimization

8.2.1 Optimizer selection

Two different types of optimizers are available for linear problems: The default is an interior-point method, and the alternatives are simplex methods. The optimizer can be selected using the parameter `mosek.iparam.optimizer`.

8.2.2 The interior-point optimizer

The purpose of this section is to provide information about the algorithm employed in MOSEK interior-point optimizer.

In order to keep the discussion simple it is assumed that MOSEK solves linear optimization problems on standard form

$$\begin{aligned} & \text{minimize} && c^T x \\ & \text{subject to} && Ax = b, \\ & && x \geq 0. \end{aligned} \tag{8.1}$$

This is in fact what happens inside MOSEK; for efficiency reasons MOSEK converts the problem to standard form before solving, then convert it back to the input form when reporting the solution.

Since it is not known beforehand whether problem (8.1) has an optimal solution, is primal infeasible or is dual infeasible, the optimization algorithm must deal with all three situations. This is the reason that MOSEK solves the so-called homogeneous model

$$\begin{aligned} Ax - b\tau &= 0, \\ A^T y + s - c\tau &= 0, \\ -c^T x + b^T y - \kappa &= 0, \\ x, s, \tau, \kappa &\geq 0, \end{aligned} \tag{8.2}$$

where y and s correspond to the dual variables in (8.1), and τ and κ are two additional scalar variables.

Note that the homogeneous model (8.2) always has solution since

$$(x, y, s, \tau, \kappa) = (0, 0, 0, 0, 0)$$

is a solution, although not a very interesting one.

Any solution

$$(x^*, y^*, s^*, \tau^*, \kappa^*)$$

to the homogeneous model (8.2) satisfies

$$x_j^* s_j^* = 0 \text{ and } \tau^* \kappa^* = 0.$$

Moreover, there is always a solution that has the property

$$\tau^* + \kappa^* > 0.$$

First, assume that $\tau^* > 0$. It follows that

$$\begin{aligned} A \frac{x^*}{\tau^*} &= b, \\ A^T \frac{y^*}{\tau^*} + \frac{s^*}{\tau^*} &= c, \\ -c^T \frac{x^*}{\tau^*} + b^T \frac{y^*}{\tau^*} &= 0, \\ x^*, s^*, \tau^*, \kappa^* &\geq 0. \end{aligned} \tag{8.3}$$

This shows that $\frac{x^*}{\tau^*}$ is a primal optimal solution and $(\frac{y^*}{\tau^*}, \frac{s^*}{\tau^*})$ is a dual optimal solution; this is reported as the optimal interior-point solution since

$$(x, y, s) = \left(\frac{x^*}{\tau^*}, \frac{y^*}{\tau^*}, \frac{s^*}{\tau^*} \right)$$

is a primal-dual optimal solution.

On other hand, if $\kappa^* > 0$ then

$$\begin{aligned} Ax^* &= 0, \\ A^T y^* + s^* &= 0, \\ -c^T x^* + b^T y^* &= \kappa^*, \\ x^*, s^*, \tau^*, \kappa^* &\geq 0. \end{aligned} \tag{8.4}$$

This implies that at least one of

$$-c^T x^* > 0 \tag{8.5}$$

or

$$b^T y^* > 0 \tag{8.6}$$

is satisfied. If (8.5) is satisfied then x^* is a certificate of dual infeasibility, whereas if (8.6) is satisfied then y^* is a certificate of dual infeasibility.

In summary, by computing an appropriate solution to the homogeneous model, all information required for a solution to the original problem is obtained. A solution to the homogeneous model can be computed using a primal-dual interior-point algorithm [10].

8.2.2.1 Interior-point termination criterion

For efficiency reasons it is not practical to solve the homogeneous model exactly. Hence, an exact optimal solution or an exact infeasibility certificate cannot be computed and a reasonable termination criterion has to be employed.

In every iteration, k , of the interior-point algorithm a trial solution

$$(x^k, y^k, s^k, \tau^k, \kappa^k)$$

to homogeneous model is generated where

$$x^k, s^k, \tau^k, \kappa^k > 0.$$

Whenever the trial solution satisfies the criterion

$$\min \left(\frac{(x^k)^T s^k + \tau^k \kappa^k}{(\tau^k)^2}, \left\| A^T \frac{y^k}{\tau^k} + \frac{s^k}{\tau^k} - c \right\|, \left\| A \frac{x^k}{\tau^k} - b \right\| \right) \leq \varepsilon_g \max \left(1, \left| \frac{c^T x^k}{\tau^k} \right| \right), \quad (8.7)$$

the interior-point optimizer is terminated and

$$\frac{(x^k, y^k, s^k)}{\tau^k}$$

is reported as the primal-dual optimal solution. The interpretation of (8.7) is that the optimizer is terminated if

- $\frac{x^k}{\tau^k}$ is approximately primal feasible,
- $\left(\frac{y^k}{\tau^k}, \frac{s^k}{\tau^k} \right)$ is approximately dual feasible, and
- the duality gap is almost zero.

On the other hand, if the trial solution satisfies

$$-\varepsilon_i c^T x^k > \frac{\|c\|}{\max(\|b\|, 1)} \|Ax^k\| \quad (8.8)$$

then the problem is declared dual infeasible and x^k is reported as a certificate of dual infeasibility. The motivation for this stopping criterion is as follows: First assume that $\|Ax^k\| = 0$; then x^k is an exact certificate of dual infeasibility. Next assume that this is not the case, i.e.

$$\|Ax^k\| > 0,$$

and define

$$\bar{x} := \varepsilon_i \frac{\max(1, \|b\|) x^k}{\|Ax^k\| \|c\|}.$$

Tolerance	Parameter name
ε_p	<code>mosek.dparam.intpnt_tol_pfeas</code>
ε_d	<code>mosek.dparam.intpnt_tol_dfeas</code>
ε_g	<code>mosek.dparam.intpnt_tol_rel_gap</code>
ε_i	<code>mosek.dparam.intpnt_tol_infeas</code>

Table 8.1: Parameters employed in termination criterion.

It is easy to verify that

$$\|A\bar{x}\| = \varepsilon_i \text{ and } -c^T \bar{x} > 1,$$

which shows \bar{x} is an approximate certificate dual infeasibility where ε_i controls the quality of the approximation. A smaller value means a better approximation.

Finally, if

$$\varepsilon_i b^T y^k \geq \frac{\|b\|}{\max(1, \|c\|)} \|A^T y^k + s^k\| \quad (8.9)$$

then y^k is reported as a certificate of primal infeasibility.

It is possible to adjust the tolerances ε_p , ε_d , ε_g and ε_i using parameters; see table 8.1 for details.

The default values of the termination tolerances are chosen such that for a majority of problems appearing in practice it is not possible to achieve much better accuracy. Therefore, tightening the tolerances usually is not worthwhile. However, an inspection of (8.7) reveals that quality of the solution is dependent on $\|b\|$ and $\|c\|$; the smaller the norms are, the better the solution accuracy.

The interior-point method as implemented by MOSEK will converge toward optimality and primal and dual feasibility at the same rate [10]. This means that if the optimizer is stopped prematurely then it is very unlikely that either the primal or dual solution is feasible. Another consequence is that in most cases all the tolerances, ε_p , ε_d and ε_g , has to be relaxed together to achieve an effect.

The basis identification discussed in section 8.2.2.2 requires an optimal solution to work well; hence basis identification should be turned off if the termination criterion is relaxed.

To conclude the discussion in this section, relaxing the termination criterion is usually not worthwhile.

8.2.2.2 Basis identification

An interior-point optimizer does not return an optimal basic solution unless the problem has a unique primal and dual optimal solution. Therefore, the interior-point optimizer has an optional post-processing step that computes an optimal basic solution starting from the optimal interior-point solution. More information about the basis identification procedure may be found in [7].

Please note that a basic solution is often more accurate than an interior-point solution.

By default MOSEK performs a basis identification. However, if a basic solution is not needed, the basis identification procedure can be turned off. The parameters

- `mosek.iparam.intpnt_basis`,

- `mosek.iparam.bi_ignore_max_iter`, and
- `mosek.iparam.bi_ignore_num_error`

controls when basis identification is performed.

8.2.2.3 The interior-point log

Below is a typical log output from the interior-point optimizer presented:

```

Optimizer - threads           : 1
Optimizer - solved problem    : the dual
Optimizer - constraints       : 2          variables           : 6
Factor    - setup time        : 0.04       order time          : 0.00
Factor    - GP order used     : no          GP order time        : 0.00
Factor    - nonzeros before factor : 3       after factor         : 3
Factor    - offending columns : 0          flops                : 1.70e+001
ITE PFEAS   DFEAS   KAP/TAU POBJ          DOBJ          MU          TIME
0  2.0e+002  2.9e+001  2.0e+002 -0.000000000e+000 -1.204741644e+003  2.0e+002  0.44
1  2.2e+001  3.1e+000  7.3e+002 -5.885951891e+003 -5.856764353e+003  2.2e+001  0.57
2  3.8e+000  5.4e-001  9.7e+001 -7.405187479e+003 -7.413054916e+003  3.8e+000  0.58
3  4.0e-002  5.7e-003  2.6e-001 -7.664507945e+003 -7.665313396e+003  4.0e-002  0.58
4  4.2e-006  6.0e-007  2.7e-005 -7.667999629e+003 -7.667999714e+003  4.2e-006  0.59
5  4.2e-010  6.0e-011  2.7e-009 -7.667999994e+003 -7.667999994e+003  4.2e-010  0.59

```

The first line displays the number of threads used by the optimizer and second line tells that the optimizer choose to solve the dual problem rather the primal problem. The next line displays the problem dimensions as seen by the optimizer, and the “Factor...” lines show various statistics. This is followed by the iteration log.

Using the same notation as in section 8.2.2 the columns of the iteration log has the following meaning:

- ITE: Iteration index.
- PFEAS: $\|Ax^k - b\tau^k\|$. The numbers in this column should converge monotonically towards to zero.
- DFEAS: $\|A^T y^k + s^k - c\tau^k\|$. The numbers in this column should converge monotonically toward to zero.
- KAP/TAU: κ^k/τ^k . If the numbers in this column converge toward zero then the problem has an optimal solution. Otherwise if the numbers converge towards infinity, the problem is primal or/and dual infeasible.
- POBJ: $c^T x^k/\tau^k$. An estimate for the primal objective value.
- DOBJ: $b^T y^k/\tau^k$. An estimate for the dual objective value.
- MU: $\frac{(x^k)^T s^k + \tau^k \kappa^k}{n+1}$. The numbers in this column should always converge monotonically to zero.
- TIME: Time spend since the optimization started.

8.2.3 The simplex based optimizer

An alternative to the interior-point optimizer is the simplex optimizer.

The simplex optimizer uses a different method that allows exploiting an initial guess for the optimal solution to reduce the solution time. Depending on the problem it may be faster or slower to use an initial guess; see section 8.2.4 for a discussion.

MOSEK provides both a primal and a dual variant of the simplex optimizer — we will return to this later.

8.2.3.1 Simplex termination criterion

The simplex optimizer terminates when it finds an optimal basic solution or an infeasibility certificate. A basic solution is optimal when it is primal and dual feasible; see (7.1) and (7.2) for a definition of the primal and dual problem. Due the fact that to computations are performed in finite precision MOSEK allows violation of primal and dual feasibility within certain tolerances. The user can control the allowed primal and dual infeasibility with the parameters `mosek.dparam.basis_tol_x` and `mosek.dparam.basis_tol_s`.

8.2.3.2 Starting from an existing solution

When using the simplex optimizer it may be possible to reuse an existing solution and thereby reduce the solution time significantly. When a simplex optimizer starts from an existing solution it is said to perform a *hot-start*. If the user is solving a sequence of optimization problems by solving the problem, making modifications, and solving again, MOSEK will hot-start automatically.

Setting the parameter `mosek.iparam.optimizer` to `mosek.optimizertype.free_simplex` instructs MOSEK to select automatically between the primal and the dual simplex optimizers. Hence, MOSEK tries to choose the best optimizer for the given problem and the available solution.

By default MOSEK uses presolve when performing a hot-start. If the optimizer only needs very few iterations to find the optimal solution it may be better to turn off the presolve.

8.2.3.3 Numerical difficulties in the simplex optimizers

Though MOSEK is designed to minimize numerical instability, completely avoiding it is impossible when working in finite precision. MOSEK counts a “numerical unexpected behavior” event inside the optimizer as a *set-back*. The user can define how many set-backs the optimizer accepts; if that number is exceeded, the optimization will be aborted. Set-backs are implemented to avoid long sequences where the optimizer tries to recover from an unstable situation.

Set-backs are, for example, repeated singularities when factorizing the basis matrix, repeated loss of feasibility, degeneracy problems (no progress in objective) and other events indicating numerical difficulties. If the simplex optimizer encounters a lot of set-backs the problem is usually badly scaled; in such a situation try to reformulate into a better scaled problem. Then, if a lot of set-backs still occur, trying one or more of the following suggestions may be worthwhile:

- Raise tolerances for allowed primal or dual feasibility: Hence, increase the value of
 - `mosek.dparam.basis_tol_x`, and
 - `mosek.dparam.basis_tol_s`.
- Raise or lower pivot tolerance: Change the `mosek.dparam.simplex_abs_tol_piv` parameter.
- Switch optimizer: Try another optimizer.
- Switch off crash: Set both `mosek.iparam.sim_primal_crash` and `mosek.iparam.sim_dual_crash` to 0.
- Experiment with other pricing strategies: Try different values for the parameters
 - `mosek.iparam.sim_primal_selection` and
 - `mosek.iparam.sim_dual_selection`.
- If you are using hot-starts, in rare cases switching off this feature may improve stability. This is controlled by the `mosek.iparam.sim_hotstart` parameter.
- Increase maximum set-backs allowed controlled by `mosek.iparam.sim_max_num_setbacks`.
- If the problem repeatedly becomes infeasible try switching off the special degeneracy handling. See the parameter `mosek.iparam.sim_degen` for details.

8.2.4 The interior-point or the simplex optimizer?

Given a linear optimization problem, which optimizer is the best: The primal simplex, the dual 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 hot-start, while simplex can take advantage of an initial solution, but is less predictable for cold-start. The interior-point optimizer is used by default.

8.2.5 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, makes it faster on average than the primal simplex optimizer. Still, it depends much on the problem structure and size.

Setting the `mosek.iparam.optimizer` parameter to `mosek.optimizertype.free_simplex` instructs MOSEK to choose which simplex optimizer to use automatically.

To summarize, if you want to know which optimizer is faster for a given problem type, you should try all the optimizers.

Alternatively, use the concurrent optimizer presented in Section 8.6.3.

8.3 Linear network optimization

8.3.1 Network flow problems

Linear optimization problems with the network flow structure as specified in section 6.1 can often be solved significantly faster with a specialized version of the simplex method [2] than with the general solvers.

MOSEK includes a network simplex solver which, on average, solves network problems 10 to 100 times faster than the standard simplex optimizers.

To use the network simplex optimizer, do the following:

- Input the network flow problem as an ordinary linear optimization problem.
- Set the parameters
 - `mosek.iparam.sim_network_detect` to 0, and
 - `mosek.iparam.optimizer` to `mosek.optimizertype.free_simplex`.
- Optimize the problem using `Task.optimize`.

MOSEK will automatically detect the network structure and apply the specialized simplex optimizer.

8.3.2 Embedded network problems

Often problems contain both large parts with network structure and some non-network constraints or variables — such problems are said to have *embedded network structure*.

If the procedure described in section 8.3.1 is applied, MOSEK will attempt to exploit this structure to speed up the optimization.

This is done heuristically by detecting the largest network embedded in the problem, solving this subproblem using the network simplex optimizer, and using the solution to hot-start a normal simplex optimizer.

The `mosek.iparam.sim_network_detect` parameter defines how large a percentage of the problem should be a network before the specialized solver is applied. In general, it is recommended to use the network optimizer only on problems containing a substantial embedded network.

If MOSEK only finds limited network structure in a problem, consider trying to switch off presolve `mosek.iparam.presolve_use` and scaling `mosek.iparam.sim_scaling`, since in rare cases it might disturb the network heuristic.

The network detection heuristic can also be called directly through `Task.netextraction`.

Parameter name	Purpose
<code>mosek.dparam.intpnt_co_tol_pfeas</code>	Controls primal feasibility
<code>mosek.dparam.intpnt_co_tol_dfeas</code>	Controls dual feasibility
<code>mosek.dparam.intpnt_co_tol_rel_gap</code>	Controls relative gap
<code>mosek.dparam.intpnt_tol_infeas</code>	Controls when the problem is declared infeasible
<code>mosek.dparam.intpnt_co_tol_mu_red</code>	Controls when the complementarity is reduced enough

Table 8.2: Parameters employed in termination criterion.

8.4 Conic optimization

8.4.1 The interior-point optimizer

For conic optimization problems only an interior-point type optimizer is available. The interior-point optimizer is an implementation of the so-called homogeneous and self-dual algorithm. For a detailed description of the algorithm, please see [6].

8.4.1.1 Interior-point termination criteria

The parameters controlling when the conic interior-point optimizer terminates are shown in Table 8.2.

8.5 Nonlinear convex optimization

8.5.1 The interior-point optimizer

For quadratic, quadratically constrained, and general convex optimization problems an interior-point type optimizer is available. The interior-point optimizer is an implementation of the homogeneous and self-dual algorithm. For a detailed description of the algorithm, please see [8, 9].

8.5.1.1 The convexity requirement

Continuous nonlinear problems are required to be convex. For quadratic problems MOSEK test this requirement before optimizing. Specifying a non-convex problem results in an error message.

The following parameters are available to control the convexity check:

- `mosek.iparam.check_convexity`: Turn convexity check on/off.
- `mosek.dparam.check_convexity_rel_tol`: Tolerance for convexity check.
- `mosek.iparam.log_check_convexity`: Turn on more log information for debugging.

8.5.1.2 The differentiability requirement

The nonlinear optimizer in MOSEK requires both first order and second order derivatives. This of course implies care should be taken when solving problems involving non-differentiable functions.

For instance, the function

$$f(x) = x^2$$

is differentiable everywhere whereas the function

$$f(x) = \sqrt{x}$$

is only differentiable for $x > 0$. In order to make sure that MOSEK evaluates the functions at points where they are differentiable, the function domains must be defined by setting appropriate variable bounds.

In general, if a variable is not ranged MOSEK will only evaluate that variable at points strictly within the bounds. Hence, imposing the bound

$$x \geq 0$$

in the case of \sqrt{x} is sufficient to guarantee that the function will only be evaluated in points where it is differentiable.

However, if a function is differentiable on closed a range, specifying the variable bounds is not sufficient. Consider the function

$$f(x) = \frac{1}{x} + \frac{1}{1-x}. \quad (8.10)$$

In this case the bounds

$$0 \leq x \leq 1$$

will not guarantee that MOSEK only evaluates the function for x between 0 and 1. To force MOSEK to strictly satisfy both bounds on ranged variables set the parameter `mosek.iparam.intpnt_starting_point` to `mosek.startpointtype.satisfy_bounds`.

For efficiency reasons it may be better to reformulate the problem than to force MOSEK to observe ranged bounds strictly. For instance, (8.10) can be reformulated as follows

$$\begin{aligned} f(x) &= \frac{1}{x} + \frac{1}{y} \\ 0 &= 1 - x - y \\ 0 &\leq x \\ 0 &\leq y. \end{aligned}$$

8.5.1.3 Interior-point termination criteria

The parameters controlling when the general convex interior-point optimizer terminates are shown in Table 8.3.

8.6 Solving problems in parallel

If a computer has multiple CPUs, or has a CPU with multiple cores, it is possible for MOSEK to take advantage of this to speed up solution times.

Parameter name	Purpose
<code>mosek.dparam.intpnt.nl_tol_pfeas</code>	Controls primal feasibility
<code>mosek.dparam.intpnt.nl_tol_dfeas</code>	Controls dual feasibility
<code>mosek.dparam.intpnt.nl_tol_rel_gap</code>	Controls relative gap
<code>mosek.dparam.intpnt_tol_infeas</code>	Controls when the problem is declared infeasible
<code>mosek.dparam.intpnt.nl_tol_mu_red</code>	Controls when the complementarity is reduced enough

Table 8.3: Parameters employed in termination criteria.

8.6.1 Thread safety

The MOSEK API is thread-safe provided that a task is only modified or accessed from one thread at any given time — accessing two separate tasks from two separate threads at the same time is safe. Sharing an environment between threads is safe.

8.6.2 The parallelized interior-point optimizer

The interior-point optimizer is capable of using multiple CPUs or cores. This implies that whenever the MOSEK interior-point optimizer solves an optimization problem, it will try to divide the work so that each CPU gets a share of the work. The user decides how many CPUs MOSEK should exploit.

It is not always possible to divide the work equally, and often parts of the computations and the coordination of the work is processed sequentially, even if several CPUs are present. Therefore, the speed-up obtained when using multiple CPUs is highly problem dependent. However, as a rule of thumb, if the problem solves very quickly, i.e. in less than 60 seconds, it is not advantageous to use the parallel option.

The `mosek.iparam.intpnt_num_threads` parameter sets the number of threads (and therefore the number of CPUs) that the interior point optimizer will use.

8.6.3 The concurrent optimizer

An alternative to the parallel interior-point optimizer is the *concurrent optimizer*. The idea of the concurrent optimizer is to run multiple optimizers on the same problem concurrently, for instance, it allows you to apply the interior-point and the dual simplex optimizers to a linear optimization problem concurrently. The concurrent optimizer terminates when the first of the applied optimizers has terminated successfully, and it reports the solution of the fastest optimizer. In that way a new optimizer has been created which essentially performs as the fastest of the interior-point and the dual simplex optimizers. Hence, the concurrent optimizer is the best one to use if there are multiple optimizers available in MOSEK for the problem and you cannot say beforehand which one will be faster.

Note in particular that any solution present in the task will also be used for hot-starting the simplex algorithms. One possible scenario would therefore be running a hot-start dual simplex in parallel with interior point, taking advantage of both the stability of the interior-point method and the ability of the simplex method to use an initial solution.

Optimizer	Associated parameter	Default priority
<code>mosek.optimizertype.intpnt</code>	<code>mosek.iparam.concurrent_priority_intpnt</code>	4
<code>mosek.optimizertype.free_simplex</code>	<code>mosek.iparam.concurrent_priority_free_simplex</code>	3
<code>mosek.optimizertype.primal_simplex</code>	<code>mosek.iparam.concurrent_priority_primal_simplex</code>	2
<code>mosek.optimizertype.dual_simplex</code>	<code>mosek.iparam.concurrent_priority_dual_simplex</code>	1

Table 8.4: Default priorities for optimizer selection in concurrent optimization.

By setting the

```
mosek.iparam.optimizer
```

parameter to

```
mosek.optimizertype.concurrent
```

the concurrent optimizer chosen.

The number of optimizers used in parallel is determined by the

```
mosek.iparam.concurrent_num_optimizers.
```

parameter. Moreover, the optimizers are selected according to a preassigned priority with optimizers having the highest priority being selected first. The default priority for each optimizer is shown in Table 8.6.3. For example, setting the `mosek.iparam.concurrent_num_optimizers` parameter to 2 tells the concurrent optimizer to apply the two optimizers with highest priorities: In the default case that means the interior-point optimizer and one of the simplex optimizers.

8.6.3.1 Concurrent optimization through the API

The following example shows how to call the concurrent optimizer through the API.

```

1  ##
2  #   Copyright: Copyright (c) 1998-2011 MOSEK ApS, Denmark. All rights reserved.
3  #
4  #   File:      concurrent1.py
5  #
6  #   Purpose: To demonstrate how to optimize in parallel using the
7  #             concurrent optimizer.
8  ##
9
10
11 import sys
12
13 import mosek
14
15 from mosek.array import array

```

```

16
17
18 # Since the actual value of Infinity is ignores, we define it solely
19 # for symbolic purposes:
20 inf = 0.0
21
22 # Define a stream printer to grab output from MOSEK
23 def streamprinter(text):
24     sys.stdout.write(text)
25     sys.stdout.flush()
26
27
28 # We might write everything directly as a script, but it looks nicer
29 # to create a function.
30 def main (args):
31     # Open MOSEK and create an environment and task
32     # Create a MOSEK environment
33     env = mosek.Env ()
34     # Attach a printer to the environment
35     env.set_Stream (mosek.streamtype.log, streamprinter)
36
37     # Create a task
38     task = env.Task(0,0)
39     # Attach a printer to the task
40     task.set_Stream (mosek.streamtype.log, streamprinter)
41
42     task.readdata(args[0])
43     task.putintparam(mosek.iparam.optimizer,
44                     mosek.optimizertype.concurrent)
45
46     task.putintparam(mosek.iparam.concurrent_num_optimizers, 2)
47
48     task.optimize()
49
50     task.solutionsummary(mosek.streamtype.msg)
51
52 # call the main function
53 try:
54     main (sys.argv[1:])
55 except mosek.Exception, (code,msg):
56     print "ERROR: %s" % str(code)
57     if msg is not None:
58         print "\t%s" % msg
59     sys.exit(1)
60 except:
61     import traceback
62     traceback.print_exc()
63     sys.exit(1)
64
65 sys.exit(0)

```

8.6.4 A more flexible concurrent optimizer

MOSEK also provides a more flexible method of concurrent optimization by using the function `Task.optimizeconcurrent`. The main advantages of this function are that it allows the calling ap-

plication to assign arbitrary values to the parameters of each tasks, and that call-back functions can be attached to each task. This may be useful in the following situation: Assume that you know the primal simplex optimizer to be the best optimizer for your problem, but that you do not know which of the available selection strategies (as defined by the `mosek.iparam.sim_primal_selection` parameter) is the best. In this case you can solve the problem with the primal simplex optimizer using several different selection strategies concurrently.

An example demonstrating the usage of the `Task.optimizeconcurrent` function is included below. The example solves a single problem using the interior-point and primal simplex optimizers in parallel.

```

1  ##
2  # Copyright: Copyright (c) 1998-2011 MOSEK ApS, Denmark. All rights reserved.
3  #
4  # File:      concurrent2.py
5  #
6  # Purpose:   To demonstrate a more flexible interface for concurrent optimization.
7  ##
8
9
10 import sys
11
12 import mosek
13
14 from mosek.array import array
15
16 # Since the actual value of Infinity is ignored, we define it solely
17 # for symbolic purposes:
18 inf = 0.0
19
20 # Define a stream printer to grab output from MOSEK
21 class streamprinter:
22     def __init__(self, prefix):
23         self.prefix = str(prefix)
24     def __call__(self, text):
25         sys.stdout.write (self.prefix + text)
26         sys.stdout.write (self.prefix + text)
27         sys.stdout.flush()
28         pass
29
30 # We might write everything directly as a script, but it looks nicer
31 # to create a function.
32 def main (args):
33     # Open MOSEK and create an environment and task
34     # Create a MOSEK environment
35     env = mosek.Env ()
36     # Attach a printer to the environment
37     env.set_Stream (mosek.streamtype.log, streamprinter("[env]"))
38
39     # Create a task
40     task = env.Task(0,0)
41     # Attach a printer to the task
42     task.set_Stream (mosek.streamtype.log, streamprinter("simplex: "))
43
44     # Create a task
45     task_list = [env.Task(0,0)]
46     # Attach a printer to the task
47     task_list[0].set_Stream(mosek.streamtype.log, streamprinter("intpnt: "))

```

```

48     task.readdata(args[0]);
49
50
51     # Assign different parameter values to each task.
52     # In this case different optimizers.
53     task.putintparam(mosek.iparam.optimizer,
54                     mosek.optimizertype.primal_simplex)
55
56     task_list[0].putintparam(mosek.iparam.optimizer,
57                             mosek.optimizertype.intpnt)
58
59
60     # Optimize task and task_list[0] in parallel.
61     # The problem data i.e. C, A, etc.
62     # is copied from task to task_list[0].
63     task.optimizeconcurrent(task_list)
64
65     task.solutionsummary(mosek.streamtype.log)
66
67
68 # call the main function
69 try:
70     main (sys.argv[1:])
71 except mosek.Exception, e:
72     print "ERROR: %s" % str(e.errno)
73     if e.msg is not None:
74         print "\t%s" % e.msg
75     sys.exit(1)
76 except:
77     import traceback
78     traceback.print_exc()
79     sys.exit(1)

```

8.7 Understanding solution quality

MOSEK will, in general, not produce an *exact* optimal solution; for efficiency reasons computations are performed in finite precision. This means that it is important to evaluate the quality of the reported solution. To evaluate the solution quality inspect the following properties:

- The solution status reported by MOSEK.
- Primal feasibility: How much the solution violates the original constraints of the problem.
- Dual feasibility: How much the dual solution violates the constraints of the dual problem.
- Duality gap: The difference between the primal and dual objective values.

Ideally, the primal and dual solutions should only violate the constraints of their respective problem *slightly* and the primal and dual objective values should be *close*. This should be evaluated in the context of the problem: How good is the data precision in the problem, and how exact a solution is required.

8.7.1 The solution summary

The solution summary is a small display generated by MOSEK that makes it easy to check the quality of the solution.

The function `Task.solutionsummary` should be used to generate solution summary.

8.7.1.1 The optimal case

The solution summary has the format

```
Problem status : PRIMAL_AND_DUAL_FEASIBLE
Solution status : OPTIMAL
Primal - objective: 5.5018458883e+03    eq. infeas.: 1.20e-12 max bound infeas.: 2.31e-14
Dual   - objective: 5.5018458883e+03    eq. infeas.: 1.15e-14 max bound infeas.: 7.11e-15
```

i.e. it shows status information, objective values and quality measures for the primal and dual solutions.

Assumeing that we are solving a linear optimization problem and referring to the problems (7.1) and (7.2), the interpretation of the solution summary is as follows:

- Problem status: The status of the problem.
- Solution status: The status of the solution.
- Primal objective: The primal objective value.
- Primal eq. infeas: $\|Ax^x - x^c\|_\infty$.
- Primal max bound infeas.: $\max(l^c - x^c; x^c - u^c; l^x - x^x; x^x - u^x; 0)$.
- Dual objective: The dual objective value.
- Dual eq. infeas: $\| -y + s_l^c - s_u^c; A^T y + s_l^x - s_u^x - c \|_\infty$.
- Dual max bound infeas.: $\max(-s_l^c; -s_u^c; -s_l^x; -s_u^x; 0)$.

In the solution summary above the solution is classified as **OPTIMAL**, meaning that the solution should be a good approximation to the true optimal solution. This seems very reasonable since the primal and dual solutions only violate their respective constraints slightly. Moreover, the duality gap is small, i.e. the primal and dual objective values are almost identical.

8.7.1.2 The primal infeasible case

For an infeasible problem the solution summary might look like this:

```
Problem status : PRIMAL_INFEASIBLE
Solution status : PRIMAL_INFEASIBLE_CER
Primal - objective: 0.0000000000e+00    eq. infeas.: 0.00e+00 max bound infeas.: 0.00e+00
Dual   - objective: 1.0000000000e+02    eq. infeas.: 0.00e+00 max bound infeas.: 0.00e+00
```

It is known that if the problem is primal infeasible then an infeasibility certificate exists, which is a solution to the problem (7.3) having a positive objective value. Note that the primal solution plays no role and only the dual solution is used to specify the certificate.

Therefore, in the primal infeasible case the solution summary should report how good the dual solution is to the problem (7.3). The interpretation of the solution summary is as follows:

- Problem status: The status of the problem.
- Solution status: The status of the solution.
- Primal objective: Should be ignored.
- Primal eq. infeas: Should be ignored.
- Primal max bound infeas.: Should be ignored.
- Dual objective: $(l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x$.
- Dual eq. infeas: $\| -y + s_l^c - s_u^c; A^T y + s_l^x - s_u^x - 0 \|_\infty$.
- Dual max bound infeas.: $\max(-s_l^c; -s_u^c; -s_l^x; -s_u^x)$.

Please note that

- any information about the primal solution should be ignored.
- the dual objective value should be strictly positive if primal problem is minimization problem. Otherwise it should be strictly negative.
- the bigger the ratio

$$\frac{(l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x}{\max(\| -y + s_l^c - s_u^c; A^T y + s_l^x - s_u^x - 0 \|_\infty, \max(-s_l^c; -s_u^c; -s_l^x; -s_u^x))}$$

is, the better the certificate is. The reason is that a certificate is a ray, and hence only the direction is important. Therefore, in principle, the certificate should be normalized before using it.

Please see Section 10.2 for more information about certificates of infeasibility.

8.7.2 Retrieving solution quality information with the API

Information about the solution quality may be retrieved in the API with the help of the following functions:

- **Task.getsolutioninf**: Obtains maximum infeasibility and primal/dual objective value.
- **Task.analyzesolution**: Print additional information about the solution, e.g basis condition number and optionally a list of violated constraints.
- **Task.getpeqi**, **Task.getdeqi**, **Task.getpbi**, **Task.getdbi**, **Task.getdcni**, **Task.getpcni**: Obtains violation of the individual constraints.

Chapter 9

The optimizer for mixed integer problems

A problem is a mixed-integer optimization problem when one or more of the variables are constrained to be integers. The integer optimizer available in MOSEK can solve integer optimization problems involving

- linear,
- quadratic and
- conic

constraints. However, a problem is not allowed to have both conic constraints and quadratic objective or constraints.

Readers unfamiliar with integer optimization are strongly recommended to consult some relevant literature, e.g. the book [24] by Wolsey is a good introduction to integer optimization.

9.1 Some notation

In general, an integer optimization problem has the form

$$\begin{aligned} z^* = & \text{minimize} && c^T x \\ & \text{subject to} && \begin{array}{lll} l^c & \leq & Ax & \leq & u^c, \\ l^x & \leq & x & \leq & u^x, \\ & & x_j \in \mathcal{Z}, & & \forall j \in \mathcal{J}, \end{array} \end{aligned} \tag{9.1}$$

where \mathcal{J} is an index set specifying which variables are integer-constrained. Frequently we talk about the continuous relaxation of an integer optimization problem defined as

$$\begin{aligned} \underline{z} = & \text{minimize} && c^T x \\ & \text{subject to} && \begin{array}{l} l^c \leq Ax \leq u^c, \\ l^x \leq x \leq u^x \end{array} \end{aligned} \quad (9.2)$$

i.e. we ignore the constraint

$$x_j \in \mathbb{Z}, \forall j \in \mathcal{J}.$$

Moreover, let \hat{x} be any feasible solution to (9.1) and define

$$\bar{z} := c^T \hat{x}.$$

It should be obvious that

$$\underline{z} \leq z^* \leq \bar{z}$$

holds. This is an important observation since if we assume that it is not possible to solve the mixed-integer optimization problem within a reasonable time frame, but that a feasible solution can be found, then the natural question is: How far is the *obtained* solution from the *optimal* solution? The answer is that no feasible solution can have an objective value smaller than \underline{z} , which implies that the obtained solution is no further away from the optimum than $\bar{z} - \underline{z}$.

9.2 An important fact about integer optimization problems

It is important to understand that in a worst-case scenario, the time required to solve integer optimization problems grows exponentially with the size of the problem. For instance, assume that a problem contains n binary variables, then the time required to solve the problem in the worst case may be proportional to 2^n . It is a simple exercise to verify that 2^n is huge even for moderate values of n .

In practice this implies that the focus should be on computing a near optimal solution quickly rather than at locating an optimal solution.

9.3 How the integer optimizer works

The process of solving an integer optimization problem can be split in three phases:

Presolve: In this phase the optimizer tries to reduce the size of the problem using preprocessing techniques. Moreover, it strengthens the continuous relaxation, if possible.

Heuristic: Using heuristics the optimizer tries to guess a good feasible solution.

Optimization: The optimal solution is located using a variant of the branch-and-cut method.

In some cases the integer optimizer may locate an optimal solution in the preprocessing stage or conclude that the problem is infeasible. Therefore, the heuristic and optimization stages may never be performed.

9.3.1 Presolve

In the preprocessing stage redundant variables and constraints are removed. The presolve stage can be turned off using the `mosek.iparam.mio_presolve_use` parameter.

9.3.2 Heuristic

Initially, the integer optimizer tries to guess a good feasible solution using different heuristics:

- First a very simple rounding heuristic is employed.
- Next, if deemed worthwhile, the *feasibility pump* heuristic is used.
- Finally, if the two previous stages did not produce a good initial solution, more sophisticated heuristics are used.

The following parameters can be used to control the effort made by the integer optimizer to find an initial feasible solution.

- `mosek.iparam.mio_heuristic_level`: Controls how sophisticated and computationally expensive a heuristic to employ.
- `mosek.dparam.mio_heuristic_time`: The minimum amount of time to spend in the heuristic search.
- `mosek.iparam.mio_feaspump_level`: Controls how aggressively the feasibility pump heuristic is used.

9.3.3 The optimization phase

This phase solves the problem using the branch and cut algorithm.

9.4 Termination criterion

In general, it is impossible to find an exact feasible and optimal solution to an integer optimization problem in a reasonable amount of time, though in many practical cases it may be possible. Therefore, the integer optimizer employs a relaxed feasibility and optimality criterion to determine when a satisfactory solution is located.

A candidate solution, i.e. a solution to (9.2), is said to be an integer feasible solution if the criterion

$$\min(|x_j| - \lfloor x_j \rfloor, \lceil x_j \rceil - |x_j|) \leq \max(\delta_1, \delta_2 |x_j|) \quad \forall j \in \mathcal{J}$$

is satisfied. Hence, such a solution is defined as a feasible solution to (9.1).

Tolerance	Parameter name
δ_1	<code>mosek.dparam.mio_tol_abs_relax_int</code>
δ_2	<code>mosek.dparam.mio_tol_rel_relax_int</code>
δ_3	<code>mosek.dparam.mio_tol_abs_gap</code>
δ_4	<code>mosek.dparam.mio_tol_rel_gap</code>
δ_5	<code>mosek.dparam.mio_near_tol_abs_gap</code>
δ_6	<code>mosek.dparam.mio_near_tol_rel_gap</code>

Table 9.1: Integer optimizer tolerances.

Parameter name	Delayed	Explanation
<code>mosek.iparam.mio_max_num_branches</code>	Yes	Maximum number of branches allowed.
<code>mosek.iparam.mio_max_num_relaxs</code>	Yes	Maximum number of relaxations allowed.
<code>mosek.iparam.mio_max_num_solutions</code>	Yes	Maximum number of feasible integer solutions allowed.

Table 9.2: Parameters affecting the termination of the integer optimizer.

Whenever the integer optimizer locates an integer feasible solution it will check if the criterion

$$\bar{z} - \underline{z} \leq \max(\delta_3, \delta_4 \max(1, |\bar{z}|))$$

is satisfied. If this is the case, the integer optimizer terminates and reports the integer feasible solution as an optimal solution. Please note that \underline{z} is a valid lower bound determined by the integer optimizer during the solution process, i.e.

$$\underline{z} \leq z^*.$$

The lower bound \underline{z} normally increases during the solution process.

The δ tolerances can be specified using parameters — see Table 9.1. If an optimal solution cannot be located within a reasonable time, it may be advantageous to employ a relaxed termination criterion after some time. Whenever the integer optimizer locates an integer feasible solution and has spent at least the number of seconds defined by the `mosek.dparam.mio_disable_term_time` parameter on solving the problem, it will check whether the criterion

$$\bar{z} - \underline{z} \leq \max(\delta_5, \delta_6 \max(1, |\bar{z}|))$$

is satisfied. If it is satisfied, the optimizer will report that the candidate solution is **near optimal** and then terminate. All δ tolerances can be adjusted using suitable parameters — see Table 9.1. In Table 9.2 some other parameters affecting the integer optimizer termination criterion are shown. Please note that if the effect of a parameter is delayed, the associated termination criterion is applied only after some time, specified by the `mosek.dparam.mio_disable_term_time` parameter.

9.5 How to speed up the solution process

As mentioned previously, in many cases it is not possible to find an optimal solution to an integer optimization problem in a reasonable amount of time. Some suggestions to reduce the solution time are:

- Relax the termination criterion: In case the run time is not acceptable, the first thing to do is to relax the termination criterion — see Section 9.4 for details.
- Specify a good initial solution: In many cases a good feasible solution is either known or easily computed using problem specific knowledge. If a good feasible solution is known, it is usually worthwhile to use this as a starting point for the integer optimizer.
- Improve the formulation: A mixed-integer optimization problem may be impossible to solve in one form and quite easy in another form. However, it is beyond the scope of this manual to discuss good formulations for mixed-integer problems. For discussions on this topic see for example [24].

9.6 Understanding solution quality

To determine the quality of the solution one should check the following:

- The solution status key returned by MOSEK.
- The *optimality gap*: A measure for how much the located solution can deviate from the optimal solution to the problem.
- Feasibility. How much the solution violates the constraints of the problem.

The *optimality gap* is a measure for how close the solution is to the optimal solution. The optimality gap is given by

$$\epsilon = |(\text{objective value of feasible solution}) - (\text{objective bound})|.$$

The objective value of the solution is guaranteed to be within ϵ of the optimal solution.

The optimality gap can be retrieved through the solution item `mosek.dinfitem.mio_obj_abs_gap`. Often it is more meaningful to look at the optimality gap normalized with the magnitude of the solution. The relative optimality gap is available in `mosek.dinfitem.mio_obj_rel_gap`.

9.6.1 Solutionsummary

The function `Task.solutionsummary` prints the most important solution information to the screen.

After a call to the optimizer the solution summary might look like this:

```
Problem status : PRIMAL_FEASIBLE
Solution status : INTEGER_OPTIMAL
Primal - objective: 1.2015000000e+06   eq. infeas.: 0.00e+00 max bound infeas.: 0.00e+00
cone infeas.: 0.00e+00 integer infeas.: 0.00e+00
```

The second line contains the solution status key. This shows how MOSEK classified the solution. In this case it is `INTEGER_OPTIMAL`, meaning that the solution is considered to be optimal within the selected tolerances.

The third line contains information relating to the solution. The first number is the primal objective function. The second and third number is the maximum infeasibility in the equality constraints and bounds respectively. The fourth and fifth number is the maximum infeasibility in the conic and integral constraints. All the numbers relating to the feasibility of the solution should be small for the solution to be valid.

9.6.2 Retrieving solution quality information with the API

Information about the solution quality may be retrieved in the API with the help of the following functions:

- `Task.getsolutioninf`: Obtains maximum infeasibility.
- `Task.analyzesolution` Print additional information about the solution. E.g basis condition number and optionally a list of violated constraints.
- `Task.getpeqi`, `Task.getdeqi`, `Task.getpbi`, `Task.getdbi`, `Task.getdcni`, `Task.getpcni`, `Task.getinti`: Obtains violation of the individual constraints.

Chapter 10

The analyzers

10.1 The problem analyzer

The problem analyzer prints a detailed survey of the model's

- linear constraints and objective
- quadratic constraints
- conic constraints
- variables

In the initial stages of model formulation the problem analyzer may be used as a quick way of verifying that the model has been built or imported correctly. In later stages it can help revealing special structures within the model that may be used to tune the optimizer's performance or to identify the causes of numerical difficulties.

The problem analyzer is run from the command line using the `-anapro` argument and produces something similar to the following (this is the `problemanalyzer`'s survey of the `aflow30a` problem from the MIPLIB 2003 collection, see Appendix A for more examples):

Analyzing the problem

Constraints		Bounds		Variables	
upper bd:	421	ranged	: all	cont:	421
fixed :	58			bin :	421

Objective, min cx		
range: min c :	0.00000	min c >0: 11.0000
distrib:	c	vars
	0	421

```

      [11, 100)      150
      [100, 500]    271
-----

Constraint matrix A has
      479 rows (constraints)
      842 columns (variables)
      2091 (0.518449%) nonzero entries (coefficients)

Row nonzeros, A_i
      range: min A_i: 2 (0.23753%)      max A_i: 34 (4.038%)
distrib:
      A_i      rows      rows%      acc%
           2      421      87.89      87.89
      [8, 15]      20      4.18      92.07
      [16, 31]     30      6.26      98.33
      [32, 34]      8      1.67     100.00

Column nonzeros, A_j
      range: min A_j: 2 (0.417537%)      max A_j: 3 (0.626305%)
distrib:
      A_j      cols      cols%      acc%
           2      435      51.66      51.66
           3      407      48.34     100.00

A nonzeros, A(ij)
      range: min |A(ij)|: 1.00000      max |A(ij)|: 100.000
distrib:
      A(ij)      coeffs
      [1, 10)      1670
      [10, 100]    421
-----

Constraint bounds, lb <= Ax <= ub
distrib:
      |b|      lbs      ubs
           0      421
      [1, 10]      58      58

Variable bounds, lb <= x <= ub
distrib:
      |b|      lbs      ubs
           0      842
      [1, 10)      421
      [10, 100]    421
-----

```

The survey is divided into six different sections, each described below. To keep the presentation short with focus on key elements the analyzer generally attempts to display information on issues relevant for the current model only: E.g., if the model does not have any conic constraints (this is the case in the example above) or any integer variables, those parts of the analysis will not appear.

10.1.1 General characteristics

The first part of the survey consists of a brief summary of the model's linear and quadratic constraints (indexed by i) and variables (indexed by j). The summary is divided into three subsections:

Constraints

upper bd: The number of upper bounded constraints, $\sum_{j=0}^{n-1} a_{ij}x_j \leq u_i^c$

lower bd: The number of lower bounded constraints, $l_i^c \leq \sum_{j=0}^{n-1} a_{ij}x_j$

ranged : The number of ranged constraints, $l_i^c \leq \sum_{j=0}^{n-1} a_{ij}x_j \leq u_i^c$

fixed : The number of fixed constraints, $l_i^c = \sum_{j=0}^{n-1} a_{ij}x_j = u_i^c$

free : The number of free constraints

Bounds

upper bd: The number of upper bounded variables, $x_j \leq u_j^x$

lower bd: The number of lower bounded variables, $l_k^x \leq x_j$

ranged : The number of ranged variables, $l_k^x \leq x_j \leq u_j^x$

fixed : The number of fixed variables, $l_k^x = x_j = u_j^x$

free : The number of free variables

Variables

cont: The number of continuous variables, $x_j \in \mathbb{R}$

bin : The number of binary variables, $x_j \in \{0, 1\}$

int : The number of general integer variables, $x_j \in \mathbb{Z}$

Only constraints, bounds and domains actually in the model will be reported on, cf. appendix A; if all entities in a section turn out to be of the same kind, the number will be replaced by **all** for brevity.

10.1.2 Objective

The second part of the survey focuses on (the linear part of) the objective, summarizing the optimization sense and the coefficients' absolute value range and distribution. The number of 0 (zero) coefficients is singled out (if any such variables are in the problem).

The range is displayed using three terms:

min |c|: The minimum absolute value among all coefficients

min |c|>0: The minimum absolute value among the nonzero coefficients

max |c|: The maximum absolute value among the coefficients

If some of these extrema turn out to be equal, the display is shortened accordingly:

- If $\min |c|$ is greater than zero, the $\min |c| > 0$ term is obsolete and will not be displayed
- If only one or two different coefficients occur this will be displayed using `all` and an explicit listing of the coefficients

The absolute value distribution is displayed as a table summarizing the numbers by orders of magnitude (with a ratio of 10). Again, the number of variables with a coefficient of 0 (if any) is singled out. Each line of the table is headed by an interval (half-open intervals including their lower bounds), and is followed by the number of variables with their objective coefficient in this interval. Intervals with no elements are skipped.

10.1.3 Linear constraints

The third part of the survey displays information on the nonzero coefficients of the linear constraint matrix.

Following a brief summary of the matrix dimensions and the number of nonzero coefficients in total, three sections provide further details on how the nonzero coefficients are distributed by row-wise count (`A_i`), by column-wise count (`A_j`), and by absolute value (`|A(ij)|`). Each section is headed by a brief display of the distribution's range (`min` and `max`), and for the row/column-wise counts the corresponding densities are displayed too (in parentheses).

The distribution tables single out three particularly interesting counts: zero, one, and two nonzeros per row/column; the remaining row/column nonzeros are displayed by orders of magnitude (ratio 2). For each interval the relative and accumulated relative counts are also displayed.

Note that constraints may have both linear and quadratic terms, but the empty rows and columns reported in this part of the survey relate to the linear terms only. If empty rows and/or columns are found in the linear constraint matrix, the problem is analyzed further in order to determine if the corresponding constraints have any quadratic terms or the corresponding variables are used in conic or quadratic constraints; cf. the last two examples of appendix A.

The distribution of the absolute values, $|A(ij)|$, is displayed just as for the objective coefficients described above.

10.1.4 Constraint and variable bounds

The fourth part of the survey displays distributions for the absolute values of the finite lower and upper bounds for both constraints and variables. The number of bounds at 0 is singled out and, otherwise, displayed by orders of magnitude (with a ratio of 10).

10.1.5 Quadratic constraints

The fifth part of the survey displays distributions for the nonzero elements in the gradient of the quadratic constraints, i.e. the nonzero row counts for the column vectors Qx . The table is similar to

the tables for the linear constraints' nonzero row and column counts described in the survey's third part.

Note: Quadratic constraints may also have a linear part, but that will be included in the linear constraints survey; this means that if a problem has one or more pure quadratic constraints, part three of the survey will report an equal number of linear constraint rows with 0 (zero) nonzeros, cf. the last example in appendix A. Likewise, variables that appear in quadratic terms only will be reported as empty columns (0 nonzeros) in the linear constraint report.

10.1.6 Conic constraints

The last part of the survey summarizes the model's conic constraints. For each of the two types of cones, quadratic and rotated quadratic, the total number of cones are reported, and the distribution of the cones' dimensions are displayed using intervals. Cone dimensions of 2, 3, and 4 are singled out.

10.2 Analyzing infeasible problems

When developing and implementing a new optimization model, the first attempts will often be either infeasible, due to specification of inconsistent constraints, or unbounded, if important constraints have been left out.

In this chapter we will

- go over an example demonstrating how to locate infeasible constraints using the MOSEK infeasibility report tool,
- discuss in more general terms which properties that may cause infeasibilities, and
- present the more formal theory of infeasible and unbounded problems.

Furthermore, chapter 11 contains a discussion on a specific method for repairing infeasibility problems where infeasibilities are caused by model parameters rather than errors in the model or the implementation.

10.2.1 Example: Primal infeasibility

A problem is said to be *primal infeasible* if no solution exists that satisfy all the constraints of the problem.

As an example of a primal infeasible problem consider the problem of minimizing the cost of transportation between a number of production plants and stores: Each plant produces a fixed number of goods, and each store has a fixed demand that must be met. Supply, demand and cost of transportation per unit are given in figure 10.1.

The problem represented in figure 10.1 is infeasible, since the total demand

$$2300 = 1100 + 200 + 500 + 500 \tag{10.1}$$

problem.

10.2.2 Locating the cause of primal infeasibility

Usually a primal infeasible problem status is caused by a mistake in formulating the problem and therefore the question arises: “What is the cause of the infeasible status?” When trying to answer this question, it is often advantageous to follow these steps:

- Remove the objective function. This does not change the infeasible status but simplifies the problem, eliminating any possibility of problems related to the objective function.
- Consider whether your problem has some necessary conditions for feasibility and examine if these are satisfied, e.g. total supply should be greater than or equal to total demand.
- Verify that coefficients and bounds are reasonably sized in your problem.

If the problem is still primal infeasible, some of the constraints must be relaxed or removed completely. The MOSEK infeasibility report (Section 10.2.4) may assist you in finding the constraints causing the infeasibility.

Possible ways of relaxing your problem include:

- Increasing (decreasing) upper (lower) bounds on variables and constraints.
- Removing suspected constraints from the problem.

Returning to the transportation example, we discover that removing the fifth constraint

$$x_{12} = 200 \tag{10.4}$$

makes the problem feasible.

10.2.3 Locating the cause of dual infeasibility

A problem may also be *dual infeasible*. In this case the primal problem is often unbounded, meaning that feasible solutions exist such that the objective tends towards infinity. An example of a dual infeasible and primal unbounded problem is:

$$\begin{array}{ll} \text{minimize} & x_1 \\ \text{subject to} & x_1 \leq 5. \end{array} \tag{10.5}$$

To resolve a dual infeasibility the primal problem must be made more restricted by

- Adding upper or lower bounds on variables or constraints.
- Removing variables.
- Changing the objective.

10.2.3.1 A cautious note

The problem

$$\begin{aligned} & \text{minimize} && 0 \\ & \text{subject to} && 0 \leq x_1, \\ & && x_j \leq x_{j+1}, \quad j = 1, \dots, n-1, \\ & && x_n \leq -1 \end{aligned} \tag{10.6}$$

is clearly infeasible. Moreover, if any one of the constraints are dropped, then the problem becomes feasible.

This illustrates the worst case scenario that all, or at least a significant portion, of the constraints are involved in the infeasibility. Hence, it may not always be easy or possible to pinpoint a few constraints which are causing the infeasibility.

10.2.4 The infeasibility report

MOSEK includes functionality for diagnosing the cause of a primal or a dual infeasibility. It can be turned on by setting the `mosek.iparam.infeas_report_auto` to `mosek.onoffkey.on`. This causes MOSEK to print a report on variables and constraints involved in the infeasibility.

The `mosek.iparam.infeas_report_level` parameter controls the amount of information presented in the infeasibility report. The default value is 1.

10.2.4.1 Example: Primal infeasibility

We will reuse the example (10.3) located in `infeas.lp`:

```
\
\ An example of an infeasible linear problem.
\
minimize
  obj: + 1 x11 + 2 x12 + 1 x13
        + 4 x21 + 2 x22 + 5 x23
        + 4 x31 + 1 x32 + 2 x33
st
  s0: + x11 + x12          <= 200
  s1: + x23 + x24          <= 1000
  s2: + x31 +x33 + x34 <= 1000
  d1: + x11 + x31          = 1100
  d2: + x12                = 200
  d3: + x23 + x33          = 500
  d4: + x24 + x34          = 500
bounds
end
```

Using the command line

```
mosek -d MSK_IPAR_INFEAS_REPORT_AUTO MSK_ON infeas.lp
```

MOSEK produces the following infeasibility report

MOSEK PRIMAL INFEASIBILITY REPORT.

Problem status: The problem is primal infeasible

The following constraints are involved in the primal infeasibility.

Index	Name	Lower bound	Upper bound	Dual lower	Dual upper
0	s0	NONE	2.000000e+002	0.000000e+000	1.000000e+000
2	s2	NONE	1.000000e+003	0.000000e+000	1.000000e+000
3	d1	1.100000e+003	1.100000e+003	1.000000e+000	0.000000e+000
4	d2	2.000000e+002	2.000000e+002	1.000000e+000	0.000000e+000

The following bound constraints are involved in the infeasibility.

Index	Name	Lower bound	Upper bound	Dual lower	Dual upper
8	x33	0.000000e+000	NONE	1.000000e+000	0.000000e+000
10	x34	0.000000e+000	NONE	1.000000e+000	0.000000e+000

The infeasibility report is divided into two sections where the first section shows which constraints that are important for the infeasibility. In this case the important constraints are the ones named **s0**, **s2**, **d1**, and **d2**. The values in the columns “Dual lower” and “Dual upper” are also useful, since a non-zero *dual lower* value for a constraint implies that the lower bound on the constraint is important for the infeasibility. Similarly, a non-zero *dual upper* value implies that the upper bound on the constraint is important for the infeasibility.

It is also possible to obtain the infeasible subproblem. The command line

```
mosek -d MSK_IPAR_INFEAS_REPORT_AUTO MSK_ON infeas.lp -info rinfeas.lp
```

produces the files `rinfeas.bas.inf.lp`. In this case the content of the file `rinfeas.bas.inf.lp` is

```
minimize
  Obj: + CFIXVAR
st
s0: + x11 + x12 <= 200
s2: + x31 + x33 + x34 <= 1e+003
d1: + x11 + x31 = 1.1e+003
d2: + x12 = 200
bounds
x11 free
x12 free
```

```

x13 free
x21 free
x22 free
x23 free
x31 free
x32 free
x24 free
CFIXVAR = 0e+000
end

```

which is an optimization problem. This problem is identical to (10.3), except that the objective and some of the constraints and bounds have been removed. Executing the command

```
mosek -d MSK_IPAR_INFEAS_REPORT_AUTO MSK_ON rinfeas.bas.inf.lp
```

demonstrates that the reduced problem is **primal infeasible**. Since the reduced problem is usually smaller than original problem, it should be easier to locate the cause of the infeasibility in this rather than in the original (10.3).

10.2.4.2 Example: Dual infeasibility

The example problem

```

maximize - 200 y1 - 1000 y2 - 1000 y3
          - 1100 y4 - 200 y5 - 500 y6
          - 500 y7
subject to
  x11: y1+y4 < 1
  x12: y1+y5 < 2
  x23: y2+y6 < 5
  x24: y2+y7 < 2
  x31: y3+y4 < 1
  x33: y3+y6 < 2
  x44: y3+y7 < 1
bounds
  y1 < 0
  y2 < 0
  y3 < 0
  y4 free
  y5 free
  y6 free
  y7 free
end

```

is dual infeasible. This can be verified by proving that

$y_1=-1, y_2=-1, y_3=0, y_4=1, y_5=1$

is a certificate of dual infeasibility. In this example the following infeasibility report is produced (slightly edited):

The following constraints are involved in the infeasibility.

Index	Name	Activity	Objective	Lower bound	Upper bound
0	x11	-1.000000e+00		NONE	1.000000e+00
4	x31	-1.000000e+00		NONE	1.000000e+00

The following variables are involved in the infeasibility.

Index	Name	Activity	Objective	Lower bound	Upper bound
3	y4	-1.000000e+00	-1.100000e+03	NONE	NONE

Interior-point solution

Problem status : DUAL_INFEASIBLE

Solution status : DUAL_INFEASIBLE_CER

Primal - objective: 1.1000000000e+03 eq. infeas.: 0.00e+00 max bound infeas.: 0.00e+00 cone infeas.: 0.00e+00

Dual - objective: 0.0000000000e+00 eq. infeas.: 0.00e+00 max bound infeas.: 0.00e+00 cone infeas.: 0.00e+00

Let x^* denote the reported primal solution. MOSEK states

- that the problem is *dual infeasible*,
- that the reported solution is a certificate of dual infeasibility, and
- that the infeasibility measure for x^* is approximately zero.

Since it was an maximization problem, this implies that

$$c^t x^* > 0. \quad (10.7)$$

For a minimization problem this inequality would have been reversed — see (10.19).

From the infeasibility report we see that the variable y_4 , and the constraints x_{11} and x_{33} are involved in the infeasibility since these appear with non-zero values in the “Activity” column.

One possible strategy to “fix” the infeasibility is to modify the problem so that the certificate of infeasibility becomes invalid. In this case we may do one the the following things:

- Put a lower bound in y_3 . This will directly invalidate the certificate of dual infeasibility.
- Increase the object coefficient of y_3 . Changing the coefficients sufficiently will invalidate the inequality (10.7) and thus the certificate.
- Put lower bounds on x_{11} or x_{31} . This will directly invalidate the certificate of infeasibility.

Please note that modifying the problem to invalidate the reported certificate does *not* imply that the problem becomes dual feasible — the infeasibility may simply “move”, resulting in a new infeasibility.

More often, the reported certificate can be used to give a hint about errors or inconsistencies in the model that produced the problem.

10.2.5 Theory concerning infeasible problems

This section discusses the theory of infeasibility certificates and how MOSEK uses a certificate to produce an infeasibility report. In general, MOSEK solves the problem

$$\begin{aligned} & \text{minimize} && c^T x + c^f \\ & \text{subject to} && \begin{array}{ccc} l^c & \leq & Ax & \leq & u^c, \\ l^x & \leq & x & \leq & u^x \end{array} \end{aligned} \quad (10.8)$$

where the corresponding dual problem is

$$\begin{aligned} & \text{maximize} && (l^c)^T s_l^c - (u^c)^T s_u^c \\ & && + (l^x)^T s_l^x - (u^x)^T s_u^x + c^f \\ & \text{subject to} && \begin{array}{rcl} A^T y + s_l^x - s_u^x & = & c, \\ -y + s_l^c - s_u^c & = & 0, \\ s_l^c, s_u^c, s_l^x, s_u^x & \geq & 0. \end{array} \end{aligned} \quad (10.9)$$

We use the convention that for any bound that is not finite, the corresponding dual variable is fixed at zero (and thus will have no influence on the dual problem). For example

$$l_j^x = -\infty \Rightarrow (s_l^x)_j = 0 \quad (10.10)$$

10.2.6 The certificate of primal infeasibility

A certificate of primal infeasibility is *any* solution to the homogenized dual problem

$$\begin{aligned} & \text{maximize} && (l^c)^T s_l^c - (u^c)^T s_u^c \\ & && + (l^x)^T s_l^x - (u^x)^T s_u^x \\ & \text{subject to} && \begin{array}{rcl} A^T y + s_l^x - s_u^x & = & 0, \\ -y + s_l^c - s_u^c & = & 0, \\ s_l^c, s_u^c, s_l^x, s_u^x & \geq & 0. \end{array} \end{aligned} \quad (10.11)$$

with a positive objective value. That is, $(s_l^{c*}, s_u^{c*}, s_l^{x*}, s_u^{x*})$ is a certificate of primal infeasibility if

$$(l^c)^T s_l^{c*} - (u^c)^T s_u^{c*} + (l^x)^T s_l^{x*} - (u^x)^T s_u^{x*} > 0 \quad (10.12)$$

and

$$\begin{aligned} A^T y + s_l^{x*} - s_u^{x*} &= 0, \\ -y + s_l^{c*} - s_u^{c*} &= 0, \\ s_l^{c*}, s_u^{c*}, s_l^{x*}, s_u^{x*} &\geq 0. \end{aligned} \quad (10.13)$$

The well-known Farkas Lemma tells us that (10.8) is infeasible if and only if a certificate of primal infeasibility exists.

Let $(s_l^{c*}, s_u^{c*}, s_l^{x*}, s_u^{x*})$ be a certificate of primal infeasibility then

$$(s_l^{c*})_i > 0 \quad ((s_u^{c*})_i > 0) \quad (10.14)$$

implies that the lower (upper) bound on the i th constraint is important for the infeasibility. Furthermore,

$$(s_l^{x*})_j > 0 \quad ((s_u^{x*})_j > 0) \quad (10.15)$$

implies that the lower (upper) bound on the j th variable is important for the infeasibility.

10.2.7 The certificate of dual infeasibility

A certificate of dual infeasibility is *any* solution to the problem

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & \bar{l}^c \leq Ax \leq \bar{u}^c, \\ & \bar{l}^x \leq x \leq \bar{u}^x \end{array} \quad (10.16)$$

with negative objective value, where we use the definitions

$$\bar{l}_i^c := \begin{cases} 0, & l_i^c > -\infty, \\ -\infty, & \text{otherwise,} \end{cases} \quad \bar{u}_i^c := \begin{cases} 0, & u_i^c < \infty, \\ \infty, & \text{otherwise,} \end{cases} \quad (10.17)$$

and

$$\bar{l}_i^x := \begin{cases} 0, & l_i^x > -\infty, \\ -\infty, & \text{otherwise,} \end{cases} \quad \text{and} \quad \bar{u}_i^x := \begin{cases} 0, & u_i^x < \infty, \\ \infty, & \text{otherwise.} \end{cases} \quad (10.18)$$

Stated differently, a certificate of dual infeasibility is any x^* such that

$$\begin{array}{lll} c^T x^* & < & 0, \\ \bar{l}^c & \leq & Ax^* \leq \bar{u}^c, \\ \bar{l}^x & \leq & x^* \leq \bar{u}^x \end{array} \quad (10.19)$$

The well-known Farkas Lemma tells us that (10.9) is infeasible if and only if a certificate of dual infeasibility exists.

Note that if x^* is a certificate of dual infeasibility then for any j such that

$$x_j^* \neq 0, \quad (10.20)$$

variable j is involved in the dual infeasibility.

Chapter 11

Primal feasibility repair

Section 10.2.2 discusses how MOSEK treats infeasible problems. In particular, it is discussed which information MOSEK returns when a problem is infeasible and how this information can be used to pinpoint the elements causing the infeasibility.

In this section we will discuss a method for repairing a primal infeasible problem by relaxing the constraints in a controlled way. For the sake of simplicity we discuss the method in the context of linear optimization. MOSEK can also repair infeasibilities in quadratic and conic optimization problems possibly having integer constrained variables. Please note that infeasibilities in nonlinear optimization problems can't be repaired using the method described below.

11.1 The main idea

Consider the linear optimization problem with m constraints and n variables

$$\begin{array}{ll} \text{minimize} & c^T x + c^f \\ \text{subject to} & l^c \leq Ax \leq u^c, \\ & l^x \leq x \leq u^x, \end{array} \quad (11.1)$$

which we assume is infeasible. Moreover, we assume that

$$(l^c)_i \leq (u^c)_i, \quad \forall i \quad (11.2)$$

and

$$(l^x)_j \leq (u^x)_j, \quad \forall j \quad (11.3)$$

because otherwise the problem (11.1) is trivially infeasible.

One way of making the problem feasible is to reduce the lower bounds and increase the upper bounds. If the change is sufficiently large the problem becomes feasible.

One obvious question is: What is the smallest change to the bounds that will make the problem feasible?

We associate a weight with each bound:

- $w_l^c \in \mathbb{R}^m$ (associated with l^c),
- $w_u^c \in \mathbb{R}^m$ (associated with u^c),
- $w_l^x \in \mathbb{R}^n$ (associated with l^x),
- $w_u^x \in \mathbb{R}^n$ (associated with u^x),

Now, the problem

$$\begin{aligned}
& \text{minimize} && p \\
& \text{subject to} && l^c \leq Ax + v_l^c - v_u^c \leq u^c, \\
& && l^x \leq x + v_l^x - v_u^x \leq u^x, \\
& && (w_l^c)^T v_l^c + (w_u^c)^T v_u^c + (w_l^x)^T v_l^x + (w_u^x)^T v_u^x - p \leq 0, \\
& && v_l^c, v_u^c, v_l^x, v_u^x \geq 0
\end{aligned} \tag{11.4}$$

minimizes the weighted sum of changes to the bounds that makes the problem feasible. The variables $(v_l^c)_i$, $(v_u^c)_i$, $(v_l^x)_i$ and $(v_u^x)_i$ are *elasticity* variables because they allow a constraint to be violated and hence add some elasticity to the problem. For instance, the elasticity variable $(v_l^c)_i$ shows how much the lower bound $(l^c)_i$ should be relaxed to make the problem feasible. Since p is minimized and

$$(w_l^c)^T v_l^c + (w_u^c)^T v_u^c + (w_l^x)^T v_l^x + (w_u^x)^T v_u^x - p \leq 0, \tag{11.5}$$

a large $(w_l^c)_i$ tends to imply that the elasticity variable $(v_l^c)_i$ will be small in an optimal solution.

The reader may want to verify that the problem (11.4) is always feasible given the assumptions (11.2) and (11.3).

Please note that if a weight is negative then the resulting problem (11.4) is unbounded.

The weights w_l^c , w_u^c , w_l^x , and w_u^x can be regarded as a costs (penalties) for violating the associated constraints. Thus a higher weight implies that higher priority is given to the satisfaction of the associated constraint.

The main idea can now be presented as follows. If you have an infeasible problem, then form the problem (11.4) and optimize it. Next inspect the optimal solution $(v_l^c)^*$, $(v_u^c)^*$, $(v_l^x)^*$, and $(v_u^x)^*$ to problem (11.4). This solution provides a suggested relaxation of the bounds that will make the problem feasible.

Assume that p^* is an optimal objective value to (11.4). An extension of the idea presented above is to solve the problem

$$\begin{aligned}
& \text{minimize} && c^T x \\
& \text{subject to} && l^c \leq Ax + v_l^c - v_u^c \leq u^c, \\
& && l^x \leq x + v_l^x - v_u^x \leq u^x, \\
& && (w_l^c)^T v_l^c + (w_u^c)^T v_u^c + (w_l^x)^T v_l^x + (w_u^x)^T v_u^x - p \leq 0, \\
& && p = p^*, \\
& && v_l^c, v_u^c, v_l^x, v_u^x \geq 0
\end{aligned} \tag{11.6}$$

which minimizes the true objective while making sure that total weighted violations of the bounds is minimal, i.e. equals to p^* .

11.2 Feasibility repair in MOSEK

MOSEK includes functionality that help you construct the problem (11.4) simply by passing a set of weights to MOSEK. This can be used for linear, quadratic, and conic optimization problems, possibly having integer constrained variables.

11.2.1 Usage of negative weights

As the problem (11.4) is presented it does not make sense to use negative weights since that makes the problem unbounded. Therefore, if the value of a weight is negative MOSEK fixes the associated elasticity variable to zero, e.g. if

$$(w_l^c)_i < 0$$

then MOSEK imposes the bound

$$(v_l^c)_i \leq 0.$$

This implies that the lower bound on the i th constraint will not be violated. (Clearly, this could also imply that the problem is infeasible so negative weight should be used with care). Associating a negative weights with a constraint tells MOSEK that the constraint should not be relaxed.

11.2.2 Automatical naming

MOSEK can automatically create a new problem of the form (11.4) starting from an existing problem by adding the elasticity variables and the extra constraints. Specifically, the variables v_l^c , v_u^c , v_l^x , v_u^x , and p are appended to existing variable vector x in their natural order. Moreover, the constraint (11.5) is appended to the constraints.

The new variables and constraints are automatically given names as follows:

- The names of the variables $(v_l^c)_i$ and $(v_u^c)_i$ are constructed from the name of the i th constraint. For instance, if the 9th original constraint is named `c9`, then by default $(v_l^c)_9$ and $(v_u^c)_9$ are given the names `L0*c9` and `UP*c9` respectively. If necessary, the character “*” can be replaced by a different string by changing the `mosek.sparam.feasrepair_name_separator` parameter.
- The additional constraints

$$l^x \leq x + v_l^x - v_u^x \leq u^x$$

are given names as follows. There is exactly one constraint per variable in the original problem, and thus the i th of these constraints is named after the i th variable in the original problem. For instance, if the first original variable is named “`x0`”, then the first of the above constraints is named “`MSK-x1`”. If necessary, the prefix “`MSK-`” can be replaced by a different string by changing the

`mosek.sparam.feasrepair_name_prefix` parameter.

- The variable p is by default given the name `WSUMVIOLVAR`, and the constraint (11.5) is given the name `WSUMVIOLCON`.

The substring “WSUMVIOL” can be replaced by a different string by changing the `mosek.sparam.feasrepair_name_wsumviol` parameter.

11.2.3 Feasibility repair using the API

The `Task.relaxprimal` function takes an existing problem as input and creates a new task containing the problem (11.4). Moreover, if requested this function can solve the problems (11.4) and (11.6) automatically.

The parameter `mosek.iparam.feasrepair_optimize` controls which problem is solved. Its value is used as follows:

- `mosek.feasrepairtype.optimize_none`: The problem (11.4) is constructed, but not solved.
- `mosek.feasrepairtype.optimize_penalty`: The problem (11.4) is constructed and solved.
- `mosek.feasrepairtype.optimize_combined`: The problem (11.6) is constructed and solved.

For further details, please see the description of the function `Task.relaxprimal` in the reference.

11.2.4 An example

Consider this example of linear optimization

$$\begin{array}{llllll}
 \text{minimize} & -10x_1 & & -9x_2, & & \\
 \text{subject to} & 7/10x_1 & + & 1x_2 & \leq & 630, \\
 & 1/2x_1 & + & 5/6x_2 & \leq & 600, \\
 & 1x_1 & + & 2/3x_2 & \leq & 708, \\
 & 1/10x_1 & + & 1/4x_2 & \leq & 135, \\
 & x_1, & & x_2 & \geq & 0. \\
 & & & & & x_2 \geq 650
 \end{array} \tag{11.7}$$

This is an infeasible problem. Suppose that we want MOSEK to suggest a modification to the bounds such that the problem becomes feasible. The following example performs this task:

```

1 #
2 #   Copyright: Copyright (c) 1998-2011 MOSEK ApS, Denmark. All rights reserved.
3 #
4 #   File:      feasrepairex1.py
5 #
6 #   Purpose:   To demonstrate how to use the MSK_relaxprimal function to
7 #             locate the cause of an infeasibility.
8 #
9 #   Syntax:   On command line
10 #            feasrepairex1 feasrepair.lp
11 #            feasrepair.lp is located in mosek\<version>\tools\examples.

```

```

12
13
14 import sys
15
16 import mosek
17
18 # If numpy is installed, use that, otherwise use the
19 # Mosek's array module.
20 try:
21     from numpy import array,zeros,ones
22 except ImportError:
23     from mosek.array import array, zeros, ones
24
25 # Since the actual value of Infinity is ignored, we define it solely
26 # for symbolic purposes:
27 inf = 0.0
28
29 # Define a stream printer to grab output from MOSEK
30 def streamprinter(text):
31     sys.stdout.write(text)
32     sys.stdout.flush()
33
34
35 def formatdarray(a):
36     r = []
37     for v in a:
38         r.append(str(v))
39     return ','.join(r)
40
41 # We might write everything directly as a script, but it looks nicer
42 # to create a function.
43 def main(inputfile):
44     # Make a MOSEK environment
45     env = mosek.Env()
46     # Attach a printer to the environment
47     env.set_Stream(mosek.streamtype.log, streamprinter)
48
49     # Create a task
50     task = env.Task(0,0)
51     # Attach a printer to the task
52     task.set_Stream(mosek.streamtype.log, streamprinter)
53
54     # Read data
55     task.readdata(inputfile)
56
57     task.putintparam(mosek.iparam.feasrepair_optimize,
58                     mosek.feasrepairtype.optimize_penalty)
59
60
61     # Relax
62     wlc = array([ 1.0, 1.0, 1.0, 1.0 ])
63     wuc = array([ 1.0, 1.0, 1.0, 1.0 ])
64     wlx = array([ 1.0, 1.0 ])
65     wux = array([ 1.0, 1.0 ])
66
67     relaxed_task = task.relaxprimal(wlc,
68                                     wuc,
69                                     wlx,

```

```

70                                     wux);
71
72     sum_violation = relaxed_task.getprimalobj (mosek.soltype.bas)
73
74     print 'lbc =', formatdarray(wlc)
75     print 'ubc =', formatdarray(wuc)
76     print 'lbx =', formatdarray(wlx)
77     print 'ubx =', formatdarray(wux)
78
79
80 # call the main function
81 try:
82     main (sys.argv[1])
83 except mosek.Exception, (code,msg):
84     print "ERROR: %s" % str(code)
85     if msg is not None:
86         print "\t%s" % msg
87     sys.exit(1)
88 except:
89     import traceback
90     traceback.print_exc()
91     sys.exit(1)
92 sys.exit(0)

```

The output from the program above is:

```

lbc = -inf,-inf,-inf,-inf
ubc = 630.0,600.0,708.0,135.0
lbx = 0.0,-inf
ubx = inf,inf

```

To make the problem feasible it is suggested increasing the upper bound on the activity of the fourth constraint from 134 to 157.5 and decreasing the lower bound on the variable x_2 to 630.

Chapter 12

Sensitivity analysis

12.1 Introduction

Given an optimization problem it is often useful to obtain information about how the optimal objective value changes when the problem parameters are perturbed. E.g, assume that a bound represents a capacity of a machine. Now, it may be possible to expand the capacity for a certain cost and hence it is worthwhile knowing what the value of additional capacity is. This is precisely the type of questions the sensitivity analysis deals with.

Analyzing how the optimal objective value changes when the problem data is changed is called sensitivity analysis.

12.2 Restrictions

Currently, sensitivity analysis is only available for continuous linear optimization problems. Moreover, MOSEK can only deal with perturbations in bounds and objective coefficients.

12.3 References

The book [13] discusses the classical sensitivity analysis in Chapter 10 whereas the book [19, Chapter 19] presents a modern introduction to sensitivity analysis. Finally, it is recommended to read the short paper [22] to avoid some of the pitfalls associated with sensitivity analysis.

12.4 Sensitivity analysis for linear problems

12.4.1 The optimal objective value function

Assume that we are given the problem

$$\begin{aligned} z(l^c, u^c, l^x, u^x, c) = & \text{minimize} && c^T x \\ & \text{subject to} && l^c \leq Ax \leq u^c, \\ & && l^x \leq x \leq u^x, \end{aligned} \quad (12.1)$$

and we want to know how the optimal objective value changes as l_i^c is perturbed. To answer this question we define the perturbed problem for l_i^c as follows

$$\begin{aligned} f_{l_i^c}(\beta) = & \text{minimize} && c^T x \\ & \text{subject to} && l^c + \beta e_i \leq Ax \leq u^c, \\ & && l^x \leq x \leq u^x, \end{aligned} \quad (12.2)$$

where e_i is the i th column of the identity matrix. The function

$$f_{l_i^c}(\beta) \quad (12.3)$$

shows the optimal objective value as a function of β . Please note that a change in β corresponds to a perturbation in l_i^c and hence (12.3) shows the optimal objective value as a function of l_i^c .

It is possible to prove that the function (12.3) is a piecewise linear and convex function, i.e. the function may look like the illustration in Figure 12.1.

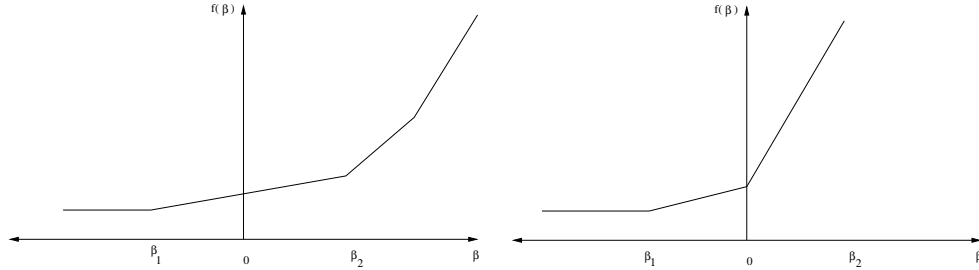


Figure 12.1: The optimal value function $f_{l_i^c}(\beta)$. Left: $\beta = 0$ is in the interior of linearity interval. Right: $\beta = 0$ is a breakpoint.

Clearly, if the function $f_{l_i^c}(\beta)$ does not change much when β is changed, then we can conclude that the optimal objective value is insensitive to changes in l_i^c . Therefore, we are interested in the rate of change in $f_{l_i^c}(\beta)$ for small changes in β — specifically the gradient

$$f'_{l_i^c}(0), \quad (12.4)$$

which is called the *shadow price* related to l_i^c . The shadow price specifies how the objective value changes for small changes in β around zero. Moreover, we are interested in the *linearity interval*

$$\beta \in [\beta_1, \beta_2] \quad (12.5)$$

for which

$$f'_{l_i^c}(\beta) = f'_{l_i^c}(0). \quad (12.6)$$

Since $f_{l_i^c}$ is not a smooth function $f'_{l_i^c}$ may not be defined at 0, as illustrated by the right example in figure 12.1. In this case we can define a left and a right shadow price and a left and a right linearity interval.

The function $f_{l_i^c}$ considered only changes in l_i^c . We can define similar functions for the remaining parameters of the z defined in (12.1) as well:

$$\begin{aligned} f_{u_i^c}(\beta) &= z(l^c, u^c + \beta e_i, l^x, u^x, c), & i = 1, \dots, m, \\ f_{l_j^x}(\beta) &= z(l^c, u^c, l^x + \beta e_j, u^x, c), & j = 1, \dots, n, \\ f_{u_j^x}(\beta) &= z(l^c, u^c, l^x, u^x + \beta e_j, c), & j = 1, \dots, n, \\ f_{c_j}(\beta) &= z(l^c, u^c, l^x, u^x, c + \beta e_j), & j = 1, \dots, n. \end{aligned} \quad (12.7)$$

Given these definitions it should be clear how linearity intervals and shadow prices are defined for the parameters u_i^c etc.

12.4.1.1 Equality constraints

In MOSEK a constraint can be specified as either an equality constraint or a ranged constraint. If constraint i is an equality constraint, we define the optimal value function for this as

$$f_{e_i^c}(\beta) = z(l^c + \beta e_i, u^c + \beta e_i, l^x, u^x, c) \quad (12.8)$$

Thus for an equality constraint the upper and the lower bounds (which are equal) are perturbed simultaneously. Therefore, MOSEK will handle sensitivity analysis differently for a ranged constraint with $l_i^c = u_i^c$ and for an equality constraint.

12.4.2 The basis type sensitivity analysis

The classical sensitivity analysis discussed in most textbooks about linear optimization, e.g. [13, Chapter 10], is based on an optimal basic solution or, equivalently, on an optimal basis. This method may produce misleading results [19, Chapter 19] but is **computationally cheap**. Therefore, and for historical reasons this method is available in MOSEK.

We will now briefly discuss the basis type sensitivity analysis. Given an optimal basic solution which provides a partition of variables into basic and non-basic variables, the basis type sensitivity analysis computes the linearity interval $[\beta_1, \beta_2]$ so that the basis remains optimal for the perturbed problem. A shadow price associated with the linearity interval is also computed. However, it is well-known that an optimal basic solution may not be unique and therefore the result depends on the optimal basic solution employed in the sensitivity analysis. This implies that the computed interval is only a subset of the largest interval for which the shadow price is constant. Furthermore, the optimal objective value function might have a breakpoint for $\beta = 0$. In this case the basis type sensitivity method will only provide a subset of either the left or the right linearity interval.

In summary, the basis type sensitivity analysis is computationally cheap but does not provide complete information. Hence, the results of the basis type sensitivity analysis should be used with care.

12.4.3 The optimal partition type sensitivity analysis

Another method for computing the complete linearity interval is called the *optimal partition type sensitivity analysis*. The main drawback of the optimal partition type sensitivity analysis is that it is computationally expensive compared to the basis type analysts. This type of sensitivity analysis is currently provided as an experimental feature in MOSEK.

Given the optimal primal and dual solutions to (12.1), i.e. x^* and $((s_l^c)^*, (s_u^c)^*, (s_l^x)^*, (s_u^x)^*)$ the optimal objective value is given by

$$z^* := c^T x^*. \quad (12.9)$$

The left and right shadow prices σ_1 and σ_2 for l_i^c are given by this pair of optimization problems:

$$\begin{aligned} \sigma_1 = & \text{minimize} && e_i^T s_l^c \\ & \text{subject to} && A^T(s_l^c - s_u^c) + s_l^x - s_u^x = c, \\ & && (l_c)^T(s_l^c) - (u_c)^T(s_u^c) + (l_x)^T(s_l^x) - (u_x)^T(s_u^x) = z^*, \\ & && s_l^c, s_u^c, s_l^x, s_u^x \geq 0 \end{aligned} \quad (12.10)$$

and

$$\begin{aligned} \sigma_2 = & \text{maximize} && e_i^T s_l^c \\ & \text{subject to} && A^T(s_l^c - s_u^c) + s_l^x - s_u^x = c, \\ & && (l_c)^T(s_l^c) - (u_c)^T(s_u^c) + (l_x)^T(s_l^x) - (u_x)^T(s_u^x) = z^*, \\ & && s_l^c, s_u^c, s_l^x, s_u^x \geq 0. \end{aligned} \quad (12.11)$$

These two optimization problems make it easy to interpret the shadow price. Indeed, if $((s_l^c)^*, (s_u^c)^*, (s_l^x)^*, (s_u^x)^*)$ is an arbitrary optimal solution then

$$(s_l^c)^*_i \in [\sigma_1, \sigma_2]. \quad (12.12)$$

Next, the linearity interval $[\beta_1, \beta_2]$ for l_i^c is computed by solving the two optimization problems

$$\begin{aligned} \beta_1 = & \text{minimize} && \beta \\ & \text{subject to} && l^c + \beta e_i \leq Ax \leq u^c, \\ & && c^T x - \sigma_1 \beta = z^*, \\ & && l^x \leq x \leq u^x, \end{aligned} \quad (12.13)$$

and

$$\begin{aligned} \beta_2 = & \text{maximize} && \beta \\ & \text{subject to} && l^c + \beta e_i \leq Ax \leq u^c, \\ & && c^T x - \sigma_2 \beta = z^*, \\ & && l^x \leq x \leq u^x. \end{aligned} \quad (12.14)$$

The linearity intervals and shadow prices for u_i^c , l_j^x , and u_j^x are computed similarly to l_i^c .

The left and right shadow prices for c_j denoted σ_1 and σ_2 respectively are computed as follows:

$$\begin{aligned} \sigma_1 = & \text{minimize} && e_j^T x \\ & \text{subject to} && l^c + \beta e_i \leq Ax \leq u^c, \\ & && c^T x = z^*, \\ & && l^x \leq x \leq u^x \end{aligned} \quad (12.15)$$

and

$$\begin{aligned} \sigma_2 = & \text{maximize} & e_j^T x \\ \text{subject to} & l^c + \beta e_i \leq Ax \leq u^c, \\ & c^T x = z^*, \\ & l^x \leq x \leq u^x. \end{aligned} \quad (12.16)$$

Once again the above two optimization problems make it easy to interpret the shadow prices. Indeed, if x^* is an arbitrary primal optimal solution, then

$$x_j^* \in [\sigma_1, \sigma_2]. \quad (12.17)$$

The linearity interval $[\beta_1, \beta_2]$ for a c_j is computed as follows:

$$\begin{aligned} \beta_1 = & \text{minimize} & \beta \\ \text{subject to} & A^T(s_l^c - s_u^c) + s_l^x - s_u^x = c + \beta e_j, \\ & (l_c)^T(s_l^c) - (u_c)^T(s_u^c) + (l_x)^T(s_l^x) - (u_x)^T(s_u^x) - \sigma_1 \beta \leq z^*, \\ & s_l^c, s_u^c, s_l^x, s_u^x \geq 0 \end{aligned} \quad (12.18)$$

and

$$\begin{aligned} \beta_2 = & \text{maximize} & \beta \\ \text{subject to} & A^T(s_l^c - s_u^c) + s_l^x - s_u^x = c + \beta e_j, \\ & (l_c)^T(s_l^c) - (u_c)^T(s_u^c) + (l_x)^T(s_l^x) - (u_x)^T(s_u^x) - \sigma_2 \beta \leq z^*, \\ & s_l^c, s_u^c, s_l^x, s_u^x \geq 0. \end{aligned} \quad (12.19)$$

12.4.4 Example: Sensitivity analysis

As an example we will use the following transportation problem. Consider the problem of minimizing the transportation cost between a number of production plants and stores. Each plant supplies a number of goods and each store has a given demand that must be met. Supply, demand and cost of transportation per unit are shown in Figure 12.2.

If we denote the number of transported goods from location i to location j by x_{ij} , the problem can be formulated as the linear optimization problem

$$\begin{aligned} \text{minimize} & 1x_{11} + 2x_{12} + 5x_{23} + 2x_{24} + 1x_{31} + 2x_{33} + 1x_{34} \end{aligned} \quad (12.20)$$

subject to

$$\begin{aligned} x_{11} + x_{12} & \leq 400, \\ x_{23} + x_{24} & \leq 1200, \\ x_{31} + x_{33} + x_{34} & \leq 1000, \\ x_{11} + x_{31} & = 800, \\ x_{12} & = 100, \\ x_{23} + x_{33} & = 500, \\ x_{24} + x_{34} & = 500, \\ x_{11}, x_{12}, x_{23}, x_{24}, x_{31}, x_{33}, x_{34} & \geq 0. \end{aligned} \quad (12.21)$$

The basis type and the optimal partition type sensitivity results for the transportation problem are shown in Table 12.1 and 12.2 respectively.

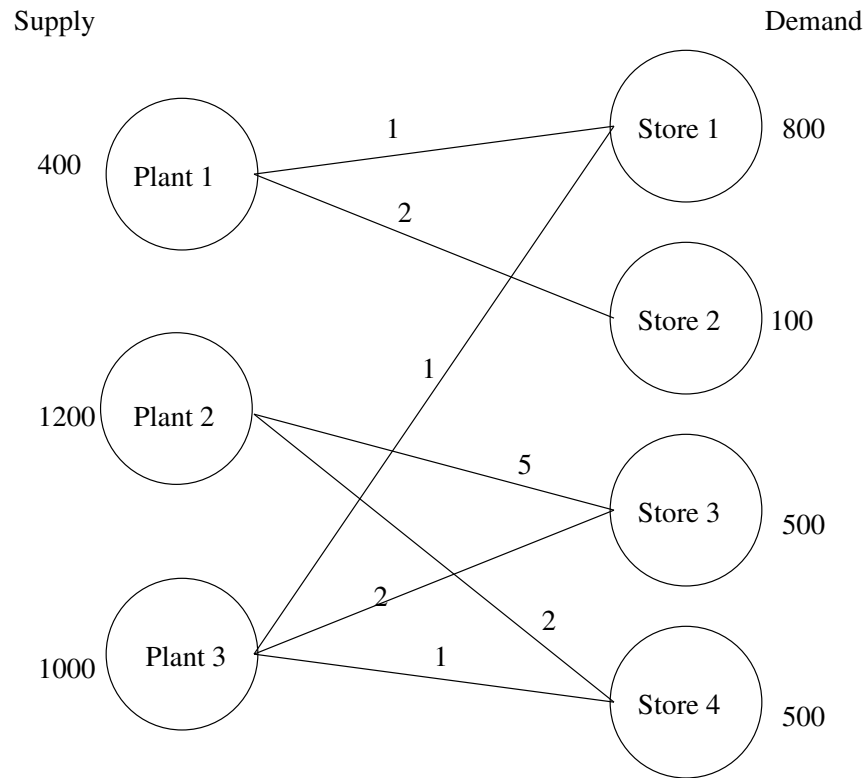


Figure 12.2: Supply, demand and cost of transportation.

Basis type					Optimal partition type				
Con.	β_1	β_2	σ_1	σ_2	Con.	β_1	β_2	σ_1	σ_2
1	-300.00	0.00	3.00	3.00	1	-300.00	500.00	3.00	1.00
2	-700.00	$+\infty$	0.00	0.00	2	-700.00	$+\infty$	-0.00	-0.00
3	-500.00	0.00	3.00	3.00	3	-500.00	500.00	3.00	1.00
4	-0.00	500.00	4.00	4.00	4	-500.00	500.00	2.00	4.00
5	-0.00	300.00	5.00	5.00	5	-100.00	300.00	3.00	5.00
6	-0.00	700.00	5.00	5.00	6	-500.00	700.00	3.00	5.00
7	-500.00	700.00	2.00	2.00	7	-500.00	700.00	2.00	2.00
Var.	β_1	β_2	σ_1	σ_2	Var.	β_1	β_2	σ_1	σ_2
x_{11}	$-\infty$	300.00	0.00	0.00	x_{11}	$-\infty$	300.00	0.00	0.00
x_{12}	$-\infty$	100.00	0.00	0.00	x_{12}	$-\infty$	100.00	0.00	0.00
x_{23}	$-\infty$	0.00	0.00	0.00	x_{23}	$-\infty$	500.00	0.00	2.00
x_{24}	$-\infty$	500.00	0.00	0.00	x_{24}	$-\infty$	500.00	0.00	0.00
x_{31}	$-\infty$	500.00	0.00	0.00	x_{31}	$-\infty$	500.00	0.00	0.00
x_{33}	$-\infty$	500.00	0.00	0.00	x_{33}	$-\infty$	500.00	0.00	0.00
x_{34}	-0.000000	500.00	2.00	2.00	x_{34}	$-\infty$	500.00	0.00	2.00

Table 12.1: Ranges and shadow prices related to bounds on constraints and variables. Left: Results for the basis type sensitivity analysis. Right: Results for the optimal partition type sensitivity analysis.

Basis type					Optimal partition type				
Var.	β_1	β_2	σ_1	σ_2	Var.	β_1	β_2	σ_1	σ_2
c_1	$-\infty$	3.00	300.00	300.00	c_1	$-\infty$	3.00	300.00	300.00
c_2	$-\infty$	∞	100.00	100.00	c_2	$-\infty$	∞	100.00	100.00
c_3	-2.00	∞	0.00	0.00	c_3	-2.00	∞	0.00	0.00
c_4	$-\infty$	2.00	500.00	500.00	c_4	$-\infty$	2.00	500.00	500.00
c_5	-3.00	∞	500.00	500.00	c_5	-3.00	∞	500.00	500.00
c_6	$-\infty$	2.00	500.00	500.00	c_6	$-\infty$	2.00	500.00	500.00
c_7	-2.00	∞	0.00	0.00	c_7	-2.00	∞	0.00	0.00

Table 12.2: Ranges and shadow prices related to the objective coefficients. Left: Results for the basis type sensitivity analysis. Right: Results for the optimal partition type sensitivity analysis.

Examining the results from the optimal partition type sensitivity analysis we see that for constraint number 1 we have $\sigma_1 \neq \sigma_2$ and $\beta_1 \neq \beta_2$. Therefore, we have a left linearity interval of $[-300, 0]$ and a right interval of $[0, 500]$. The corresponding left and right shadow prices are 3 and 1 respectively. This implies that if the upper bound on constraint 1 increases by

$$\beta \in [0, \beta_1] = [0, 500] \quad (12.22)$$

then the optimal objective value will decrease by the value

$$\sigma_2 \beta = 1\beta. \quad (12.23)$$

Correspondingly, if the upper bound on constraint 1 is decreased by

$$\beta \in [0, 300] \quad (12.24)$$

then the optimal objective value will increase by the value

$$\sigma_1 \beta = 3\beta. \quad (12.25)$$

12.5 Sensitivity analysis from the MOSEK API

MOSEK provides the functions `Task.primalsensitivity` and `Task.dualsensitivity` for performing sensitivity analysis. The code below gives an example of its use.

Example code from:

mosek/6/tools/examp/sensitivity.py

```

1  ##
2  #
3  #   Copyright: Copyright (c) 1998-2011 MOSEK ApS, Denmark. All rights reserved.
4  #
5  #   File:      sensitivity.cs
6  #
7  #   Purpose:   To demonstrate how to perform sensitivity
8  #   analysis from the API on a small problem:
9  #
10 #   minimize
11 #
12 #   obj: +1 x11 + 2 x12 + 5 x23 + 2 x24 + 1 x31 + 2 x33 + 1 x34
13 #   st
14 #   c1:    + x11 + x12                                <= 400
15 #   c2:          + x23 + x24                            <= 1200
16 #   c3:          + x31 + x33 + x34                       <= 1000
17 #   c4:    + x11                                + x31      = 800
18 #   c5:          + x12                                = 100
19 #   c6:          + x23                                + x33    = 500
20 #   c7:          + x24                                + x34    = 500
21 #
22 #   The example uses basis type sensitivity analysis.
23 ##

```

```

24
25
26 import sys
27
28 import mosek
29 # If numpy is installed, use that, otherwise use the
30 # Mosek's array module.
31 try:
32     from numpy import array,zeros,ones
33 except ImportError:
34     from mosek.array import array, zeros, ones
35
36
37 # Since the actual value of Infinity is ignores, we define it solely
38 # for symbolic purposes:
39 inf = 0.0
40
41 # Define a stream printer to grab output from MOSEK
42 def streamprinter(text):
43     sys.stdout.write(text)
44     sys.stdout.flush()
45
46
47
48 # We might write everything directly as a script, but it looks nicer
49 # to create a function.
50 def main ():
51     # Create a MOSEK environment
52     env = mosek.Env ()
53     # Attach a printer to the environment
54     env.set_Stream (mosek.streamtype.log, streamprinter)
55
56     # Create a task
57     task = env.Task(0,0)
58     # Attach a printer to the task
59     task.set_Stream (mosek.streamtype.log, streamprinter)
60
61     # Set up data
62
63     bkc = [ mosek.boundkey.up,mosek.boundkey.up,
64             mosek.boundkey.up,mosek.boundkey.fx,
65             mosek.boundkey.fx,mosek.boundkey.fx,
66             mosek.boundkey.fx ]
67     blc = [ -inf,  -inf,  -inf,  800., 100., 500., 500. ]
68     buc = [ 400., 1200., 1000., 800., 100., 500., 500. ]
69
70     bkc = [ mosek.boundkey.lo,mosek.boundkey.lo,
71             mosek.boundkey.lo,mosek.boundkey.lo,
72             mosek.boundkey.lo,mosek.boundkey.lo,
73             mosek.boundkey.lo ]
74     c   = [ 1.0,2.0,5.0,2.0,1.0,2.0,1.0 ]
75     blx = [ 0.0,0.0,0.0,0.0,0.0,0.0,0.0 ]
76     bux = [ inf,inf,inf,inf,inf,inf,inf ]
77
78     ptrb = [ 0,2,4,6, 8,10,12 ]
79     ptre = [ 2,4,6,8,10,12,14 ]
80     sub  = [ 0,3,0,4,1,5,1,6,2,3,2,5,2,6 ]
81

```

```

82 val = [ 1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,
83         1.0,1.0,1.0,1.0,1.0,1.0,1.0 ]
84
85 numcon = len(bkc)
86 numvar = len(bkx)
87 numanz = len(val)
88
89 # Input linear data
90 task.inputdata(numcon,numvar,
91               c,0.0,
92               ptrb,ptre, sub, val,
93               bkc, blc, buc,
94               bkx, blx, bux)
95
96 # Set objective sense
97 task.putobjsense(mosek.objsense.minimize)
98
99 # Optimize
100 task.optimize();
101
102 # Analyze upper bound on c1 and the equality constraint on c4
103 subi = [ 0, 3 ]
104 marki = [ mosek.mark.up, mosek.mark.up ]
105
106 # Analyze lower bound on the variables x12 and x31
107 subj = [ 1, 4 ]
108 markj = [ mosek.mark.lo, mosek.mark.lo ]
109
110 leftpricei = zeros(2,float)
111 rightpricei = zeros(2,float)
112 leftrangei = zeros(2,float)
113 rightrangei = zeros(2,float)
114 leftpricej = zeros(2,float)
115 rightpricej = zeros(2,float)
116 leftrangej = zeros(2,float)
117 rightrangej = zeros(2,float)
118
119 task.primalsensitivity( subi,
120                        marki,
121                        subj,
122                        markj,
123                        leftpricei,
124                        rightpricei,
125                        leftrangei,
126                        rightrangei,
127                        leftpricej,
128                        rightpricej,
129                        leftrangej,
130                        rightrangej)
131
132 print 'Results from sensitivity analysis on bounds:'
133 print '\tleftprice | rightprice | leftrange | rightrange '
134 print 'For constraints:'
135
136 for i in range(2):
137     print '%t%10f %10f %10f %10f' % (leftpricei[i],
138                                       rightpricei[i],
139                                       leftrangei[i],

```

```

140                                     rightrangei[i])
141
142     print 'For variables:'
143     for i in range(2):
144         print '\t%10f    %10f    %10f    %10f' % (leftpricej[i],
145                                                    rightpricej[i],
146                                                    leftrangej[i],
147                                                    rightrangej[i])
148
149
150     leftprice  = zeros(2,float)
151     rightprice = zeros(2,float)
152     leftrange  = zeros(2,float)
153     rightrange = zeros(2,float)
154     subc       = array([ 2, 5 ])
155
156     task.dualsensitivity( subc,
157                          leftprice,
158                          rightprice,
159                          leftrange,
160                          rightrange)
161
162     print 'Results from sensitivity analysis on objective coefficients:'
163
164     for i in range(2):
165         print '\t%10f    %10f    %10f    %10f' % (leftprice[i],
166                                                    rightprice[i],
167                                                    leftrange[i],
168                                                    rightrange[i])
169
170     return None
171
172 # call the main function
173 try:
174     main ()
175 except mosek.Exception, e:
176     print "ERROR: %s" % str(e.errno)
177     if e.msg is not None:
178         print "\t%s" % e.msg
179     sys.exit(1)
180 except:
181     import traceback
182     traceback.print_exc()
183     sys.exit(1)

```

12.6 Sensitivity analysis with the command line tool

A sensitivity analysis can be performed with the MOSEK command line tool using the command

```
mosek myproblem.mps -sen sensitivity.ssp
```

where `sensitivity.ssp` is a file in the format described in the next section. The `ssp` file describes which parts of the problem the sensitivity analysis should be performed on.

By default results are written to a file named `myproblem.sen`. If necessary, this filename can be changed by setting the

`MSK_SPAR_SENSITIVITY_RES_FILE_NAME`

parameter. By default a basis type sensitivity analysis is performed. However, the type of sensitivity analysis (basis or optimal partition) can be changed by setting the parameter

`MSK_IPAR_SENSITIVITY_TYPE`

appropriately. Following values are accepted for this parameter:

- `MSK_SENSITIVITY_TYPE_BASIS`
- `MSK_SENSITIVITY_TYPE_OPTIMAL_PARTITION`

It is also possible to use the command line

```
mosek myproblem.mps -d MSK_IPAR_SENSITIVITY_ALL MSK_ON
```

in which case a sensitivity analysis on all the parameters is performed.

12.6.1 Sensitivity analysis specification file

MOSEK employs an MPS like file format to specify on which model parameters the sensitivity analysis should be performed. As the optimal partition type sensitivity analysis can be computationally expensive it is important to limit the sensitivity analysis.

```
* A comment
BOUNDS CONSTRAINTS
  U|L|LU [cname1]
  U|L|LU [cname2]-[cname3]
BOUNDS VARIABLES
  U|L|LU [vname1]
  U|L|LU [vname2]-[vname3]
OBJECTIVE VARIABLES
  [vname1]
  [vname2]-[vname3]
```

Figure 12.3: The sensitivity analysis file format.

The format of the sensitivity specification file is shown in figure 12.3, where capitalized names are keywords, and names in brackets are names of the constraints and variables to be included in the analysis.

The sensitivity specification file has three sections, i.e.

- **BOUNDS CONSTRAINTS:** Specifies on which bounds on constraints the sensitivity analysis should be performed.

- **BOUNDS VARIABLES:** Specifies on which bounds on variables the sensitivity analysis should be performed.
- **OBJECTIVE VARIABLES:** Specifies on which objective coefficients the sensitivity analysis should be performed.

A line in the body of a section must begin with a whitespace. In the BOUNDS sections one of the keys L, U, and LU must appear next. These keys specify whether the sensitivity analysis is performed on the lower bound, on the upper bound, or on both the lower and the upper bound respectively. Next, a single constraint (variable) or range of constraints (variables) is specified.

Recall from Section 12.4.1.1 that equality constraints are handled in a special way. Sensitivity analysis of an equality constraint can be specified with either L, U, or LU, all indicating the same, namely that upper and lower bounds (which are equal) are perturbed simultaneously.

As an example consider

```
BOUNDS CONSTRAINTS
  L  "cons1"
  U  "cons2"
  LU "cons3"-"cons6"
```

which requests that sensitivity analysis is performed on the lower bound of the constraint named `cons1`, on the upper bound of the constraint named `cons2`, and on both lower and upper bound on the constraints named `cons3` to `cons6`.

It is allowed to use indexes instead of names, for instance

```
BOUNDS CONSTRAINTS
  L  "cons1"
  U  2
  LU 3 - 6
```

The character “*” indicates that the line contains a comment and is ignored.

12.6.2 Example: Sensitivity analysis from command line

As an example consider the `sensitivity.ssp` file shown in Figure 12.4.

The command

```
mosek transport.lp -sen sensitivity.ssp -d MSK_IPAR_SENSITIVITY_TYPE MSK_SENSITIVITY_TYPE_BASIS
```

produces the `transport.sen` file shown below.

BOUNDS CONSTRAINTS						
INDEX	NAME	BOUND	LEFTRANGE	RIGHTRANGE	LEFTPRICE	RIGHTPRICE
0	c1	UP	-6.574875e-18	5.000000e+02	1.000000e+00	1.000000e+00
2	c3	UP	-6.574875e-18	5.000000e+02	1.000000e+00	1.000000e+00

```

* Comment 1

BOUNDS CONSTRAINTS
U "c1"      * Analyze upper bound for constraint named c1
U 2        * Analyze upper bound for the second constraint
U 3-5      * Analyze upper bound for constraint number 3 to number 5

BOUNDS VARIABLES
L 2-4      * This section specifies which bounds on variables should be analyzed
L "x11"

OBJECTIVE VARIABLES
"x11"      * This section specifies which objective coefficients should be analyzed
2

```

Figure 12.4: Example of the sensitivity file format.

3	c4	FIX	-5.000000e+02	6.574875e-18	2.000000e+00	2.000000e+00
4	c5	FIX	-1.000000e+02	6.574875e-18	3.000000e+00	3.000000e+00
5	c6	FIX	-5.000000e+02	6.574875e-18	3.000000e+00	3.000000e+00

BOUNDS VARIABLES						
INDEX	NAME	BOUND	LEFTRANGE	RIGHTRANGE	LEFTPRICE	RIGHTPRICE
2	x23	LO	-6.574875e-18	5.000000e+02	2.000000e+00	2.000000e+00
3	x24	LO	-inf	5.000000e+02	0.000000e+00	0.000000e+00
4	x31	LO	-inf	5.000000e+02	0.000000e+00	0.000000e+00
0	x11	LO	-inf	3.000000e+02	0.000000e+00	0.000000e+00

OBJECTIVE VARIABLES						
INDEX	NAME	LEFTRANGE	RIGHTRANGE	LEFTPRICE	RIGHTPRICE	
0	x11	-inf	1.000000e+00	3.000000e+02	3.000000e+02	
2	x23	-2.000000e+00	+inf	0.000000e+00	0.000000e+00	

12.6.3 Controlling log output

Setting the parameter

MSK_IPAR_LOG_SENSITIVITY

to 1 or 0 (default) controls whether or not the results from sensitivity calculations are printed to the message stream.

The parameter

MSK_IPAR_LOG_SENSITIVITY_OPT

controls the amount of debug information on internal calculations from the sensitivity analysis.

Chapter 13

Usage guidelines

The purpose of this chapter is to present some general guidelines to follow when using MOSEK.

13.1 Verifying the results

The main purpose of MOSEK is to solve optimization problems and therefore the most fundamental question to be asked is whether the solution reported by MOSEK is a solution to the desired optimization problem.

There can be several reasons why it might be not case. The most prominent reasons are:

- A wrong problem. The problem inputted to MOSEK is simply not the right problem, i.e. some of the data may have been corrupted or the model has been incorrectly built.
- Numerical issues. The problem is badly scaled or otherwise badly posed.
- Other reasons. E.g. not enough memory or an explicit user request to stop.

The first step in verifying that MOSEK reports the expected solution is to inspect the solution summary generated by MOSEK; see section 8.7 for details. The solution summary provides information about

- the problem and solution statuses,
- objective value and infeasibility measures for the primal solution, and
- objective value and infeasibility measures for the dual solution, where applicable.

By inspecting the solution summary it can be verified that MOSEK produces a feasible solution, and, in the continuous case, the optimality can be checked using the dual solution. Furthermore, the problem itself can be inspected using the problem analyzer discussed in section 10.1.

If the summary reports conflicting information (e.g. a solution status that does not match the actual solution), or the cause for terminating the solver before a solution was found cannot be traced back to

the reasons stated above, it may be caused by a bug in the solver; in this case, please contact MOSEK support.

13.1.1 Verifying primal feasibility

If it has been verified that MOSEK solves the problem correctly but the solution is still not as expected, next step is to verify that the primal solution satisfies all the constraints. Hence, using the original problem it must be determined whether the solution satisfies all the required constraints in the model. For instance assume that the problem has the constraints

$$\begin{aligned} x_1 + 2x_2 + x_3 &\leq 1, \\ x_1, x_2, x_3 &\geq 0 \end{aligned}$$

and MOSEK reports the optimal solution

$$x_1 = x_2 = x_3 = 1.$$

Then clearly the solution violates the constraints. The most likely explanation is that the model does not match the problem entered into MOSEK, for instance

$$x_1 - 2x_2 + x_3 \leq 1$$

may have been inputted instead of

$$x_1 + 2x_2 + x_3 \leq 1.$$

A good way to debug such an issue is to dump the problem to OPF file and check whether the violated constraint has been specified correctly.

13.1.2 Verifying optimality

Verifying that a feasible solution is optimal can be harder. However, for continuous problems¹ optimality can be verified using a dual solution. Normally, MOSEK will report a dual solution; if that is feasible and has the same objective value as the primal solution, then the primal solution must be optimal.

An alternative method is to find another primal solution that has better objective value than the one reported to MOSEK. If that is possible then either the problem is badly posed or there is a bug in MOSEK.

13.2 Turn on logging

While developing a new application it is recommended to turn on logging, so that error and diagnostics messages are displayed.

See example in section 5.2 for instructions on turning log output on. You should also always cache and handle any exceptions thrown by MOSEK.

More log information can be obtained by modifying one or more of the parameters:

¹A problem without any integer constraints.

- `mosek.iparam.log`,
- `mosek.iparam.log_intpnt`,
- `mosek.iparam.log_mio`,
- `mosek.iparam.log_cut_second_opt`,
- `mosek.iparam.log_sim`, and
- `mosek.iparam.log_sim_minor`.

By default MOSEK will reduce the amount of log information after the first optimization on a given task. To get full log output on subsequent optimizations set:

```
mosek.iparam.log_cut_second_opt 0
```

13.3 Turn on data checking

In the development phase it is useful to use the parameter setting

```
mosek.iparam.data_check mosek.onoffkey.on
```

which forces MOSEK to check the input data. For instance, MOSEK looks for NaNs in double values and outputs a warning if any are found.

13.4 Debugging an optimization task

If something is wrong with a problem or a solution, one option is to output the problem to an OPF file and inspect it by hand. Use the `Task.writedata` function to write a task to a file immediately before optimizing, for example as follows:

This will write the problem in `task` to the file `taskdump.opf`. Inspecting the text file `taskdump.opf` may reveal what is wrong in the problem setup.

13.5 Important API limitations

13.5.1 Thread safety

The MOSEK API is thread safe in the sense that any number of threads may use it simultaneously. However, the individual tasks and environments may *only* be accessed from at most one thread at a time.

13.6 Bug reporting

If you think MOSEK is solving your problem incorrectly, please contact MOSEK support at support@mosek.com providing a detailed description of the problem. MOSEK support may ask for the task file which is produced as follows

```
task.Task.writedata("taskfile.mbt")
task.Task.optimize()
```

The task data will then be written to the `taskfile.mbt` file in binary form which is very useful when reproducing a problem.

Chapter 14

API reference

This chapter lists all functionality in the MOSEK Python API.

14.1 API Functionality

Functions in the interface grouped by functionality.

14.1.1 Analyzing the problem and associated data

Analyzing the problem and associated data.

mosek.Task.analyzeproblem (page 188)

Analyze the data of a task.

mosek.Task.analyzesolution (page 188)

Print information related to the quality of the solution.

14.1.2 Reading and writing data files

Reading and writing data files.

mosek.Task.readbranchpriorities (page 241)

Reads branching priority data from a file.

mosek.Task.readdata (page 241)

Reads problem data from a file.

mosek.Task.readparamfile (page 241)

Reads a parameter file.

- mosek.Task.readsolution** (page 241)
Reads a solution from a file.
- mosek.Task.writebranchpriorities** (page 248)
Writes branching priority data to a file.
- mosek.Task.writedata** (page 248)
Writes problem data to a file.
- mosek.Task.writeparamfile** (page 249)
Writes all the parameters to a parameter file.
- mosek.Task.writesolution** (page 249)
Write a solution to a file.

14.1.3 Solutions

Obtain or define a solution.

- mosek.Task.deletesolution** (page 191)
Undefines a solution and frees the memory it uses.
- mosek.Task.getdbi** (page 198)
Obtains the dual bound infeasibility.
- mosek.Task.getdcni** (page 198)
Obtains the dual cone infeasibility.
- mosek.Task.getdeqi** (page 199)
Obtains the dual equation infeasibility.
- mosek.Task.getdualobj** (page 199)
Obtains the dual objective value.
- mosek.Task.getinti** (page 200)
Obtains the primal equation infeasibility.
- mosek.Task.getpbi** (page 206)
Obtains the primal bound infeasibility.
- mosek.Task.getpcni** (page 207)
Obtains the primal cone infeasibility.
- mosek.Task.getpeqi** (page 207)
Obtains the primal equation infeasibility.
- mosek.Task.getprimalobj** (page 207)
Obtains the primal objective value.
- mosek.Task.getreducedcosts** (page 210)
Obtains the difference of (slx-sux) for a sequence of variables.

- mosek.Task.getsolution** (page 210)
Obtains the complete solution.
- mosek.Task.getsolutioni** (page 212)
Obtains the solution for a single constraint or variable.
- mosek.Task.getsolutioninf** (page 212)
Obtains information about a solution.
- mosek.Task.getsolutionslice** (page 214)
Obtains a slice of the solution.
- mosek.Task.getsolutionstatus** (page 215)
Obtains information about the problem and solution statuses.
- mosek.Task.getsolutionstatuskeyslice** (page 215)
Obtains a slice of the solution status keys.
- mosek.Task.makesolutionstatusunknown** (page 219)
Sets the solution status to unknown.
- mosek.Task.optimizersummary** (page 223)
Prints a short summary with optimizer statistics for last optimization.
- mosek.Task.putsolution** (page 237)
Inserts a solution.
- mosek.Task.putsolutioni** (page 238)
Sets the primal and dual solution information for a single constraint or variable.
- mosek.Task.readsolution** (page 241)
Reads a solution from a file.
- mosek.Task.solutiondef** (page 246)
Checks whether a solution is defined.
- mosek.Task.solutionsummary** (page 246)
Prints a short summary of the current solutions.
- mosek.Task.undefsolution** (page 248)
Undefines a solution.

14.1.4 Memory allocation and deallocation

Memory allocation and deallocation.

- mosek.Task.checkmem** (page 190)
Checks the memory allocated by the task.
- mosek.Task.getmemusage** (page 202)
Obtains information about the amount of memory used by a task.

14.1.5 Changing problem specification

Input or change problem specification.

mosek.Task.append (page 188)

Appends a number of variables or constraints to the optimization task.

mosek.Task.appendcone (page 189)

Appends a new cone constraint to the problem.

mosek.Env.checkoutlicense (page 174)

Check out a license feature from the license server ahead of time.

mosek.Task.chgbound (page 191)

Changes the bounds for one constraint or variable.

mosek.Task.commitchanges (page 191)

Commits all cached problem changes.

mosek.Task.inputdata (page 218)

Input the linear part of an optimization task in one function call.

mosek.Task.putaij (page 226)

Changes a single value in the linear coefficient matrix.

mosek.Task.putaijlist (page 226)

Changes one or more coefficients in the linear constraint matrix.

mosek.Task.putavec (page 227)

Replaces all elements in one row or column of the linear coefficient matrix.

mosek.Task.putaveclist (page 228)

Replaces all elements in one or more rows or columns in the linear constraint matrix by new values.

mosek.Task.putbound (page 228)

Changes the bound for either one constraint or one variable.

mosek.Task.putboundlist (page 229)

Changes the bounds of constraints or variables.

mosek.Task.putboundslice (page 229)

Modifies bounds.

mosek.Task.putcfix (page 230)

Replaces the fixed term in the objective.

mosek.Task.putcj (page 230)

Modifies one linear coefficient in the objective.

mosek.Task.putclist (page 230)

Modifies a part of the linear objective coefficients.

mosek.Task.putcone (page 231)

Replaces a conic constraint.

mosek.Task.putobjsense (page 234)

Sets the objective sense.

mosek.Task.putqcon (page 235)

Replaces all quadratic terms in constraints.

mosek.Task.putqconk (page 235)

Replaces all quadratic terms in a single constraint.

mosek.Task.putqobj (page 236)

Replaces all quadratic terms in the objective.

mosek.Task.putqobjij (page 237)

Replaces one coefficient in the quadratic term in the objective.

mosek.Task.putvartype (page 240)

Sets the variable type of one variable.

mosek.Task.putvartypelist (page 240)

Sets the variable type for one or more variables.

14.1.6 Delete problem elements (variables,constraints,cones)

Functionality for deleting problem elements such as variables, constraints or cones.

mosek.Task.remove (page 244)

The function removes a number of constraints or variables.

mosek.Task.removecone (page 244)

Removes a conic constraint from the problem.

14.1.7 Add problem elements (variables,constraints,cones)

Functionality for adding problem elements such as variables, constraints or cones.

mosek.Task.append (page 188)

Appends a number of variables or constraints to the optimization task.

mosek.Task.appendcone (page 189)

Appends a new cone constraint to the problem.

14.1.8 Problem inspection

Functionality for inspecting the problem specification (A, Q , bounds, objective e.t.c).

mosek.Task.getaij (page 192)

Obtains a single coefficient in linear constraint matrix.

mosek.Task.getaslice (page 193)

Obtains a sequence of rows or columns from the coefficient matrix.

mosek.Task.getaslicetrip (page 194)

Obtains a sequence of rows or columns from the coefficient matrix in triplet format.

mosek.Task.getavec (page 195)

Obtains one row or column of the linear constraint matrix.

mosek.Task.getavecnumnz (page 195)

Obtains the number of non-zero elements in one row or column of the linear constraint matrix

mosek.Task.getbound (page 195)

Obtains bound information for one constraint or variable.

mosek.Task.getboundslice (page 196)

Obtains bounds information for a sequence of variables or constraints.

mosek.Task.getc (page 196)

Obtains all objective coefficients.

mosek.Task.getcfix (page 196)

Obtains the fixed term in the objective.

mosek.Task.getcone (page 197)

Obtains a conic constraint.

mosek.Task.getconeinfo (page 197)

Obtains information about a conic constraint.

mosek.Task.getcslice (page 197)

Obtains a sequence of coefficients from the objective.

mosek.Task.getintpntnumthreads (page 201)

Obtains the number of threads used by the interior-point optimizer.

mosek.Task.getnumanz (page 204)

Obtains the number of non-zeros in the coefficient matrix.

mosek.Task.getnumanz64 (page 204)

Obtains the number of non-zeros in the coefficient matrix.

mosek.Task.getnumcon (page 204)

Obtains the number of constraints.

- mosek.Task.getnumcone** (page 204)
Obtains the number of cones.
- mosek.Task.getnumconemem** (page 204)
Obtains the number of members in a cone.
- mosek.Task.getnumintvar** (page 204)
Obtains the number of integer-constrained variables.
- mosek.Task.getnumqconknz** (page 205)
Obtains the number of non-zero quadratic terms in a constraint.
- mosek.Task.getnumqconknz64** (page 205)
Obtains the number of non-zero quadratic terms in a constraint.
- mosek.Task.getnumqobjnz** (page 205)
Obtains the number of non-zero quadratic terms in the objective.
- mosek.Task.getnumqobjnz64** (page 205)
Obtains the number of non-zero quadratic terms in the objective.
- mosek.Task.getnumvar** (page 205)
Obtains the number of variables.
- mosek.Task.getobjsense** (page 206)
Gets the objective sense.
- mosek.Task.getprobtype** (page 208)
Obtains the problem type.
- mosek.Task.getqconk** (page 208)
Obtains all the quadratic terms in a constraint.
- mosek.Task.getqconk64** (page 208)
Obtains all the quadratic terms in a constraint.
- mosek.Task.getqobj** (page 209)
Obtains all the quadratic terms in the objective.
- mosek.Task.getqobj64** (page 209)
Obtains all the quadratic terms in the objective.
- mosek.Task.getqobjij** (page 210)
Obtains one coefficient from the quadratic term of the objective
- mosek.Task.getvartype** (page 217)
Gets the variable type of one variable.
- mosek.Task.getvartypelist** (page 217)
Obtains the variable type for one or more variables.

14.1.9 Conic constraints

Functionality related to conic terms in the problem.

mosek.Task.appendcone (page 189)
Appends a new cone constraint to the problem.

mosek.Task.getcone (page 197)
Obtains a conic constraint.

mosek.Task.getconeinfo (page 197)
Obtains information about a conic constraint.

mosek.Task.getnumcone (page 204)
Obtains the number of cones.

mosek.Task.putcone (page 231)
Replaces a conic constraint.

mosek.Task.removecone (page 244)
Removes a conic constraint from the problem.

14.1.10 Bounds

Functionality related to changing or inspecting bounds on variables or constraints.

mosek.Task.chgbound (page 191)
Changes the bounds for one constraint or variable.

mosek.Task.getbound (page 195)
Obtains bound information for one constraint or variable.

mosek.Task.getboundslice (page 196)
Obtains bounds information for a sequence of variables or constraints.

mosek.Task.putbound (page 228)
Changes the bound for either one constraint or one variable.

mosek.Task.putboundlist (page 229)
Changes the bounds of constraints or variables.

mosek.Task.putboundslice (page 229)
Modifies bounds.

14.1.11 Output stream functions

Output stream functions.

- mosek.Env.echointro** (page 174)
Prints an intro to message stream.
- mosek.Env.linkfiletoostream** (page 175)
Directs all output from a stream to a file.
- mosek.Task.linkfiletoostream** (page 219)
Directs all output from a task stream to a file.
- mosek.Task.optimizersummary** (page 223)
Prints a short summary with optimizer statistics for last optimization.
- mosek.Task.printdata** (page 225)
Prints a part of the problem data to a stream.
- mosek.Task.printparam** (page 226)
Prints the current parameter settings.
- mosek.Task.readsummary** (page 242)
Prints information about last file read.
- mosek.Task.solutionsummary** (page 246)
Prints a short summary of the current solutions.

14.1.12 Objective function

Change or inspect objective function.

- mosek.Task.getc** (page 196)
Obtains all objective coefficients.
- mosek.Task.getcfix** (page 196)
Obtains the fixed term in the objective.
- mosek.Task.getcslice** (page 197)
Obtains a sequence of coefficients from the objective.
- mosek.Task.getdualobj** (page 199)
Obtains the dual objective value.
- mosek.Task.getnumqobjnz** (page 205)
Obtains the number of non-zero quadratic terms in the objective.
- mosek.Task.getnumqobjnz64** (page 205)
Obtains the number of non-zero quadratic terms in the objective.
- mosek.Task.getobjname** (page 206)
Obtains the name assigned to the objective function.
- mosek.Task.getobjsense** (page 206)
Gets the objective sense.

- mosek.Task.getprimalobj** (page 207)
Obtains the primal objective value.
- mosek.Task.getqobj** (page 209)
Obtains all the quadratic terms in the objective.
- mosek.Task.getqobj64** (page 209)
Obtains all the quadratic terms in the objective.
- mosek.Task.getqobjij** (page 210)
Obtains one coefficient from the quadratic term of the objective
- mosek.Task.putcfix** (page 230)
Replaces the fixed term in the objective.
- mosek.Task.putcj** (page 230)
Modifies one linear coefficient in the objective.
- mosek.Task.putclist** (page 230)
Modifies a part of the linear objective coefficients.
- mosek.Task.putobjsense** (page 234)
Sets the objective sense.
- mosek.Task.putqobj** (page 236)
Replaces all quadratic terms in the objective.
- mosek.Task.putqobjij** (page 237)
Replaces one coefficient in the quadratic term in the objective.

14.1.13 Optimizer statistics

Inspect statistics from the optimizer.

- mosek.Task.getdouninf** (page 199)
Obtains a double information item.
- mosek.Task.getintinf** (page 200)
Obtains an integer information item.
- mosek.Task.getlintinf** (page 201)
Obtains an integer information item.

14.1.14 Parameters (set/get)

Setting and inspecting solver parameters.

- mosek.Task.getdouparam** (page 199)
Obtains a double parameter.

- mosek.Task.getintparam** (page 201)
Obtains an integer parameter.
- mosek.Task.getnumparam** (page 205)
Obtains the number of parameters of a given type.
- mosek.Task.isdouparname** (page 218)
Checks a double parameter name.
- mosek.Task.isintparname** (page 219)
Checks an integer parameter name.
- mosek.Task.isstrparname** (page 219)
Checks a string parameter name.
- mosek.Task.putdouparam** (page 231)
Sets a double parameter.
- mosek.Task.putintparam** (page 231)
Sets an integer parameter.
- mosek.Task.putnadouparam** (page 233)
Sets a double parameter.
- mosek.Task.putnaintparam** (page 233)
Sets an integer parameter.
- mosek.Task.putnastrparam** (page 234)
Sets a string parameter.
- mosek.Task.putparam** (page 234)
Modifies the value of parameter.
- mosek.Task.putstrparam** (page 239)
Sets a string parameter.
- mosek.Task.setdefaults** (page 246)
Resets all parameters values.

14.1.15 Naming

Functionality related to naming.

- mosek.Task.getconname** (page 197)
Obtains a name of a constraint.
- mosek.Task.getname** (page 202)
Obtains the name of a cone, a variable or a constraint.
- mosek.Task.getname** (page 203)
Obtains the name of a cone, a variable or a constraint.

mosek.Task.getnameindex (page 203)

Checks whether a name has been assigned and returns the index corresponding to the name.

mosek.Task.getnamelen (page 203)

Obtains the length of a problem item name.

mosek.Task.getobjname (page 206)

Obtains the name assigned to the objective function.

mosek.Task.gettaskname64 (page 216)

Obtains the task name.

mosek.Task.getvarname (page 217)

Obtains a name of a variable.

mosek.Task.putname (page 233)

Assigns a name to a problem item.

mosek.Task.putobjname (page 234)

Assigns a new name to the objective.

mosek.Task.puttaskname (page 239)

Assigns a new name to the task.

14.1.16 Preallocating space for problem data

Functionality related to preallocating space for problem data.

mosek.Task.getmaxnumanz (page 201)

Obtains number of preallocated non-zeros in the linear constraint matrix.

mosek.Task.getmaxnumcon (page 201)

Obtains the number of preallocated constraints in the optimization task.

mosek.Task.getmaxnumcone (page 202)

Obtains the number of preallocated cones in the optimization task.

mosek.Task.getmaxnumqnz (page 202)

Obtains the number of preallocated non-zeros for all quadratic terms in objective and constraints.

mosek.Task.getmaxnumvar (page 202)

Obtains the maximum number variables allowed.

mosek.Task.putmaxnumanz (page 232)

The function changes the size of the preallocated storage for linear coefficients.

mosek.Task.putmaxnumcon (page 232)

Sets the number of preallocated constraints in the optimization task.

mosek.Task.putmaxnumcone (page 232)

Sets the number of preallocated conic constraints in the optimization task.

mosek.Task.putmaxnumqnz (page 232)

Changes the size of the preallocated storage for quadratic terms.

mosek.Task.putmaxnumvar (page 233)

Sets the number of preallocated variables in the optimization task.

14.1.17 Integer variables

Functionality related to integer variables.

mosek.Task.getnumintvar (page 204)

Obtains the number of integer-constrained variables.

mosek.Task.getvarbranchdir (page 216)

Obtains the branching direction for a variable.

mosek.Task.getvarbranchpri (page 216)

Obtains the branching priority for a variable.

mosek.Task.getvartype (page 217)

Gets the variable type of one variable.

mosek.Task.getvartypelist (page 217)

Obtains the variable type for one or more variables.

mosek.Task.putvarbranchorder (page 240)

Assigns a branching priority and direction to a variable.

mosek.Task.putvartype (page 240)

Sets the variable type of one variable.

mosek.Task.putvartypelist (page 240)

Sets the variable type for one or more variables.

14.1.18 Quadratic terms

Functionality related to quadratic terms.

mosek.Task.getqconk (page 208)

Obtains all the quadratic terms in a constraint.

mosek.Task.getqconk64 (page 208)

Obtains all the quadratic terms in a constraint.

mosek.Task.getqobj (page 209)

Obtains all the quadratic terms in the objective.

mosek.Task.getqobj64 (page 209)

Obtains all the quadratic terms in the objective.

- mosek.Task.getqobjjj** (page 210)
Obtains one coefficient from the quadratic term of the objective
- mosek.Task.putqcon** (page 235)
Replaces all quadratic terms in constraints.
- mosek.Task.putqconk** (page 235)
Replaces all quadratic terms in a single constraint.
- mosek.Task.putqobjj** (page 236)
Replaces all quadratic terms in the objective.
- mosek.Task.putqobjjj** (page 237)
Replaces one coefficient in the quadratic term in the objective.

14.1.19 Diagnosing infeasibility

Functions for diagnosing infeasibility.

- mosek.Task.getinfeasiblesubproblem** (page 200)
Obtains an infeasible sub problem.
- mosek.Task.relaxprimal** (page 242)
Creates a problem that finds the minimal change to the bounds that makes an infeasible problem feasible.

14.1.20 Optimization

Functions for optimization.

- mosek.Task.checkdata** (page 190)
Checks data of the task.
- mosek.Task.netoptimize** (page 221)
Optimizes a pure network flow problem.
- mosek.Task.optimizeconcurrent** (page 222)
Optimize a given task with several optimizers concurrently.
- mosek.Task.optimize** (page 223)
Optimizes the problem.

14.1.21 Network optimization

Functions for network optimization.

mosek.Task.netextraction (page 219)

Finds embedded network structure.

mosek.Task.netoptimize (page 221)

Optimizes a pure network flow problem.

14.1.22 Sensitivity analysis

Functions for sensitivity analysis.

mosek.Task.dualsensitivity (page 192)

Performs sensitivity analysis on objective coefficients.

mosek.Task.primalsensitivity (page 223)

Perform sensitivity analysis on bounds.

mosek.Task.sensitivityreport (page 245)

Creates a sensitivity report.

14.1.23 Testing data validity

Functions for testing data validity.

mosek.Task.checkconvexity (page 190)

Checks if a quadratic optimization problem is convex.

14.1.24 Solving with the basis

Functions for solving linear systems with the basis matrix.

mosek.Task.basiscond (page 190)

Computes conditioning information for the basis matrix.

mosek.Task.initbasissolve (page 217)

Prepare a task for basis solver.

mosek.Task.solvewithbasis (page 246)

Solve a linear equation system involving a basis matrix.

14.1.25 Initialization of environment

Creation and initialization of environment.

mosek.Env.checkinlicense (page 174)

Check in a license feature from the license server ahead of time.

mosek.Env.init (page 175)

Initialize a MOSEK environment.

mosek.Env.putlicensedefaults (page 176)

Set defaults used by the license manager.

14.1.26 Change A

Change elements in the coefficient (A) matrix.

mosek.Env.checkoutlicense (page 174)

Check out a license feature from the license server ahead of time.

mosek.Task.commitchanges (page 191)

Commits all cached problem changes.

mosek.Task.putaij (page 226)

Changes a single value in the linear coefficient matrix.

mosek.Task.putaijlist (page 226)

Changes one or more coefficients in the linear constraint matrix.

mosek.Task.putavec (page 227)

Replaces all elements in one row or column of the linear coefficient matrix.

mosek.Task.putaveclist (page 228)

Replaces all elements in one or more rows or columns in the linear constraint matrix by new values.

14.2 Class `mosek.ArrayLengthException`

Derived from:

`__builtin__.Exception`

Description:

This exception is raised is an input or output array was shorter than required.

14.3 Class `mosek.Env`

Description:

A Mosek Environment

14.3.1 Constructors

- `mosek.Env`

Syntax:

`Env ()`

Create a MOSEK environment object.

Description `mosek.Env`

Syntax:

`Env (str dbgfile)`

Create a MOSEK environment object.

Arguments:

`dbgfile` A file which will be used to log memory debugging information from MOSEK

14.3.2 Methods

- `mosek.Env.checkinlicense` 174
Check in a license feature from the license server ahead of time.
- `mosek.Env.checkoutlicense` 174
Check out a license feature from the license server ahead of time.
- `mosek.Env.echointro` 174
Prints an intro to message stream.
- `mosek.Env.getversion` 174
Obtains MOSEK version information.
- `mosek.Env.init` 175
Initialize a MOSEK environment.
- `mosek.Env.linkfiletoostream` 175
Directs all output from a stream to a file.
- `mosek.Env.putcpudefaults` 175
Set defaults default CPU type and cache sizes.
- `mosek.Env.putdllpath` 175
Sets the path to the DLL/shared libraries that MOSEK is loading.
- `mosek.Env.putkeepdlls` 176
Controls whether explicitly loaded DLLs should be kept.
- `mosek.Env.putlicensedefaults` 176
Set defaults used by the license manager.

- `mosek.Env.set_Stream` 176
Attach a stream call-back handler.

- `mosek.Env.checkinlicense`

Syntax:

`checkinlicense (mosek.feature feature)`

feature (input) Feature to check in to the license system.

Description: Check in a license feature to the license server. By default all licenses consumed by functions using a single environment is kept checked out for the lifetime of the MOSEK environment. This function checks in a given license feature to the license server immediately. If the given license feature is not checked out or 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.

- `mosek.Env.checkoutlicense`

Syntax:

`checkoutlicense (mosek.feature feature)`

feature (input) Feature to check out from the license system.

Description: Check out a license feature from the license server. Normally the required license features will be automatically checked out the first time it is needed by the function `Task.optimize`. This function can be used to check out one or more features ahead of time.

The license will remain checked out for the lifetime of the MOSEK environment or until the function `Env.checkinlicense` is called.

If a given feature is already checked out when this function is called, only one feature will be checked out from the license server.

- `mosek.Env.echointro`

Syntax:

`echointro (int longver)`

longver (input) If non-zero, then the intro is slightly longer.

Description: Prints an intro to message stream.

- `mosek.Env.getversion`

Syntax:

`major,minor,build,revision = getversion ()`

major (output) Major version number.

minor (output) Minor version number.

build (**output**) Build number.
 revision (**output**) Revision number.

Description: Obtains MOSEK version information.

- mosek.Env.init

Syntax:

```
init ()
```

Description: This function initializes the MOSEK environment. Among other things the license server will be contacted. Error messages from the license manager can be captured by linking to the environment message stream before calling this function.

- mosek.Env.linkfiletostream

Syntax:

```
linkfiletostream (
    mosek.streamtype whichstream,
    str filename,
    int append)
```

whichstream (input) Index of the stream.

filename (input) Sends all output from the stream defined by **whichstream** to the file given by **filename**.

append (input) If this argument is non-zero, the output is appended to the file.

Description: Directs all output from a stream to a file.

- mosek.Env.putcpudefaults

Syntax:

```
putcpudefaults (
    mosek.cputype cputype,
    int size1,
    int size2)
```

cputype (input) The CPU ID.

size1 (input) Size of the L1 cache.

size2 (input) Size of the L2 cache.

Description: Sets default CPU type and cache sizes. This function should be called before **Env.initenv**.

- mosek.Env.putdllpath

Syntax:

```
putdllpath (str dllpath)
```

dllpath (input) A path to where the MOSEK dynamic link/shared libraries are located. If **dllpath** is NULL, then MOSEK assumes that the operating system can locate the libraries.

Description: Sets the path to the DLL/shared libraries that MOSEK are loading. If needed, then it should normally be called before **Env.initenv**.

- **mosek.Env.putkeepdlls**

Syntax:

```
putkeepdlls (int keepdlls)
```

keepdlls (input) Controls whether explicitly loaded DLLs should be kept.

Description: Controls whether explicitly loaded DLLs should be kept when they no longer are in use.

- **mosek.Env.putlicensedefaults**

Syntax:

```
putlicensedefaults (
    str licensefile,
    array(int) licensebuf,
    int licwait,
    int licdebug)
```

licensefile (input) Either NULL or the path to a valid MOSEK license file.

licensebuf (input) This is the license string authorizing the use of MOSEK in the runtime version of MOSEK. Therefore, most frequently this string is a NULL pointer.

licwait (input) If this argument is non-zero, then MOSEK will wait for a license if no license is available. Moreover, **licwait-1** is the number of milliseconds to wait between each check for an available license.

licdebug (input) If this argument is non-zero, then MOSEK will print debug info regarding the license checkout.

Description: Sets default values for the license manager. This function should be called before **Env.initenv**.

- **mosek.Env.set_Stream**

Syntax:

```
set_Stream (mosek.streamtype whichstream)
```

whichstream Index of the stream.

Description: Attach a stream call-back handler.

14.4 Class mosek.Error

Derived from:

`mosek.Exception`

Description:

This is an exception class representing MOSEK errors.

14.4.1 Constructors

- `mosek.Error`

Syntax:

```
Error (mosek.rescode code)
```

Construct an error from a MOSEK error code.

Description:

code The MOSEK response code to create the exception from.

- `mosek.Error`

Syntax:

```
Error (
    mosek.rescode code,
    str msg)
```

Construct an error from a MOSEK error code and a message.

Description:

code The MOSEK response code to create the exception from.

msg A message describing the error situation.

14.5 Class mosek.Exception

Derived from:

`__builtin__.Exception`

Description:

This is the base class for exceptions based on MOSEK response codes.

14.5.1 Constructors

- `mosek.Exception`

Syntax:

```
Exception (mosek.rescode code)
```

Construct an exception from a MOSEK error code.

Description:

`code` The MOSEK response code to create the exception from.

14.6 Class `mosek.Task`

Description:

A Mosek Optimization task

14.6.1 Constructors

- `mosek.Task`

Syntax:

```
Task (
    int maxnumcon,
    int maxnumvar)
```

Create a new MOSEK task and reserve space for constraints and variables. Please note that it is perfectly legal to specify 0 constraint or 0 variables: The values may be specified later with `Task.putmaxnumcon` and `Task.putmaxnumvar`. Even without doing so, the task will automatically resize when exceeding the maximum, but if this happens often, there will be some overhead when resizing.

Description:

`maxnumcon` Initially reserve space for this many constraints.

`maxnumvar` Initially reserve space for this many variables.

14.6.2 Attributes

- `mosek.Progress` Progress (write only) Progress call-back object

14.6.3 Methods

- `mosek.Task.analyzeproblem` 188
Analyze the data of a task.
- `mosek.Task.analyzesolution` 188
Print information related to the quality of the solution.
- `mosek.Task.append` 188
Appends a number of variables or constraints to the optimization task.
- `mosek.Task.appendcone` 189
Appends a new cone constraint to the problem.
- `mosek.Task.basiscond` 190
Computes conditioning information for the basis matrix.
- `mosek.Task.checkconvexity` 190
Checks if a quadratic optimization problem is convex.
- `mosek.Task.checkdata` 190
Checks data of the task.
- `mosek.Task.checkmem` 190
Checks the memory allocated by the task.
- `mosek.Task.chgbound` 191
Changes the bounds for one constraint or variable.
- `mosek.Task.commitchanges` 191
Commits all cached problem changes.
- `mosek.Task.deletesolution` 191
Undefines a solution and frees the memory it uses.
- `mosek.Task.dualsensitivity` 192
Performs sensitivity analysis on objective coefficients.
- `mosek.Task.getaij` 192
Obtains a single coefficient in linear constraint matrix.
- `mosek.Task.getapiecenumnz` 193
Obtains the number non-zeros in a rectangular piece of the linear constraint matrix.
- `mosek.Task.getaslice` 193
Obtains a sequence of rows or columns from the coefficient matrix.
- `mosek.Task.getaslicenumnz` 194
Obtains the number of non-zeros in a slice of rows or columns of the coefficient matrix.
- `mosek.Task.getaslicetrip` 194
Obtains a sequence of rows or columns from the coefficient matrix in triplet format.

• <code>mosek.Task.getavec</code>	195
Obtains one row or column of the linear constraint matrix.	
• <code>mosek.Task.getavecnumnz</code>	195
Obtains the number of non-zero elements in one row or column of the linear constraint matrix	
• <code>mosek.Task.getbound</code>	195
Obtains bound information for one constraint or variable.	
• <code>mosek.Task.getboundslice</code>	196
Obtains bounds information for a sequence of variables or constraints.	
• <code>mosek.Task.getc</code>	196
Obtains all objective coefficients.	
• <code>mosek.Task.getcfix</code>	196
Obtains the fixed term in the objective.	
• <code>mosek.Task.getcone</code>	197
Obtains a conic constraint.	
• <code>mosek.Task.getconeinfo</code>	197
Obtains information about a conic constraint.	
• <code>mosek.Task.getconname</code>	197
Obtains a name of a constraint.	
• <code>mosek.Task.getcslice</code>	197
Obtains a sequence of coefficients from the objective.	
• <code>mosek.Task.getdbi</code>	198
Obtains the dual bound infeasibility.	
• <code>mosek.Task.getdcni</code>	198
Obtains the dual cone infeasibility.	
• <code>mosek.Task.getdeqi</code>	199
Obtains the dual equation infeasibility.	
• <code>mosek.Task.getdouinf</code>	199
Obtains a double information item.	
• <code>mosek.Task.getdoupam</code>	199
Obtains a double parameter.	
• <code>mosek.Task.getdualobj</code>	199
Obtains the dual objective value.	
• <code>mosek.Task.getinfeasiblesubproblem</code>	200
Obtains an infeasible sub problem.	
• <code>mosek.Task.getinti</code>	200
Obtains the primal equation infeasibility.	

- `mosek.Task.getintinf` 200
Obtains an integer information item.
- `mosek.Task.getintparam` 201
Obtains an integer parameter.
- `mosek.Task.getintpntnumthreads` 201
Obtains the number of threads used by the interior-point optimizer.
- `mosek.Task.getlintinf` 201
Obtains an integer information item.
- `mosek.Task.getmaxnumanz` 201
Obtains number of preallocated non-zeros in the linear constraint matrix.
- `mosek.Task.getmaxnumcon` 201
Obtains the number of preallocated constraints in the optimization task.
- `mosek.Task.getmaxnumcone` 202
Obtains the number of preallocated cones in the optimization task.
- `mosek.Task.getmaxnumqnz` 202
Obtains the number of preallocated non-zeros for all quadratic terms in objective and constraints.
- `mosek.Task.getmaxnumvar` 202
Obtains the maximum number variables allowed.
- `mosek.Task.getmemusage` 202
Obtains information about the amount of memory used by a task.
- `mosek.Task.getname` 202
Obtains the name of a cone, a variable or a constraint.
- `mosek.Task.getname` 203
Obtains the name of a cone, a variable or a constraint.
- `mosek.Task.getnameindex` 203
Checks whether a name has been assigned and returns the index corresponding to the name.
- `mosek.Task.getnamelen` 203
Obtains the length of a problem item name.
- `mosek.Task.getnumanz` 204
Obtains the number of non-zeros in the coefficient matrix.
- `mosek.Task.getnumanz64` 204
Obtains the number of non-zeros in the coefficient matrix.
- `mosek.Task.getnumcon` 204
Obtains the number of constraints.
- `mosek.Task.getnumcone` 204
Obtains the number of cones.

• <code>mosek.Task.getnumconmem</code>	204
Obtains the number of members in a cone.	
• <code>mosek.Task.getnumintvar</code>	204
Obtains the number of integer-constrained variables.	
• <code>mosek.Task.getnumparam</code>	205
Obtains the number of parameters of a given type.	
• <code>mosek.Task.getnumqconknz</code>	205
Obtains the number of non-zero quadratic terms in a constraint.	
• <code>mosek.Task.getnumqconknz64</code>	205
Obtains the number of non-zero quadratic terms in a constraint.	
• <code>mosek.Task.getnumqobjnz</code>	205
Obtains the number of non-zero quadratic terms in the objective.	
• <code>mosek.Task.getnumqobjnz64</code>	205
Obtains the number of non-zero quadratic terms in the objective.	
• <code>mosek.Task.getnumvar</code>	205
Obtains the number of variables.	
• <code>mosek.Task.getobjname</code>	206
Obtains the name assigned to the objective function.	
• <code>mosek.Task.getobjsense</code>	206
Gets the objective sense.	
• <code>mosek.Task.getpbi</code>	206
Obtains the primal bound infeasibility.	
• <code>mosek.Task.getpcni</code>	207
Obtains the primal cone infeasibility.	
• <code>mosek.Task.getpeqi</code>	207
Obtains the primal equation infeasibility.	
• <code>mosek.Task.getprimalobj</code>	207
Obtains the primal objective value.	
• <code>mosek.Task.getprobtype</code>	208
Obtains the problem type.	
• <code>mosek.Task.getqconk</code>	208
Obtains all the quadratic terms in a constraint.	
• <code>mosek.Task.getqconk64</code>	208
Obtains all the quadratic terms in a constraint.	
• <code>mosek.Task.getqobj</code>	209
Obtains all the quadratic terms in the objective.	

• <code>mosek.Task.getqobj64</code>	209
Obtains all the quadratic terms in the objective.	
• <code>mosek.Task.getqobjij</code>	210
Obtains one coefficient from the quadratic term of the objective	
• <code>mosek.Task.getreducedcosts</code>	210
Obtains the difference of (slx-sux) for a sequence of variables.	
• <code>mosek.Task.getsolution</code>	210
Obtains the complete solution.	
• <code>mosek.Task.getsolutioni</code>	212
Obtains the solution for a single constraint or variable.	
• <code>mosek.Task.getsolutioninf</code>	212
Obtains information about a solution.	
• <code>mosek.Task.getsolutionslice</code>	214
Obtains a slice of the solution.	
• <code>mosek.Task.getsolutionstatus</code>	215
Obtains information about the problem and solution statuses.	
• <code>mosek.Task.getsolutionstatuskeyslice</code>	215
Obtains a slice of the solution status keys.	
• <code>mosek.Task.gettaskname64</code>	216
Obtains the task name.	
• <code>mosek.Task.getvarbranchdir</code>	216
Obtains the branching direction for a variable.	
• <code>mosek.Task.getvarbranchpri</code>	216
Obtains the branching priority for a variable.	
• <code>mosek.Task.getvarname</code>	217
Obtains a name of a variable.	
• <code>mosek.Task.getvartype</code>	217
Gets the variable type of one variable.	
• <code>mosek.Task.getvartypelist</code>	217
Obtains the variable type for one or more variables.	
• <code>mosek.Task.initbasissolve</code>	217
Prepare a task for basis solver.	
• <code>mosek.Task.inputdata</code>	218
Input the linear part of an optimization task in one function call.	
• <code>mosek.Task.isdoupname</code>	218
Checks a double parameter name.	

• <code>mosek.Task.isintparname</code>	219
Checks an integer parameter name.	
• <code>mosek.Task.isstrparname</code>	219
Checks a string parameter name.	
• <code>mosek.Task.linkfiletostream</code>	219
Directs all output from a task stream to a file.	
• <code>mosek.Task.makesolutionstatusunknown</code>	219
Sets the solution status to unknown.	
• <code>mosek.Task.netextraction</code>	219
Finds embedded network structure.	
• <code>mosek.Task.netoptimize</code>	221
Optimizes a pure network flow problem.	
• <code>mosek.Task.optimizeconcurrent</code>	222
Optimize a given task with several optimizers concurrently.	
• <code>mosek.Task.optimizersummary</code>	223
Prints a short summary with optimizer statistics for last optimization.	
• <code>mosek.Task.optimize</code>	223
Optimizes the problem.	
• <code>mosek.Task.primalsensitivity</code>	223
Perform sensitivity analysis on bounds.	
• <code>mosek.Task.printdata</code>	225
Prints a part of the problem data to a stream.	
• <code>mosek.Task.printparam</code>	226
Prints the current parameter settings.	
• <code>mosek.Task.putaij</code>	226
Changes a single value in the linear coefficient matrix.	
• <code>mosek.Task.putaijlist</code>	226
Changes one or more coefficients in the linear constraint matrix.	
• <code>mosek.Task.putavec</code>	227
Replaces all elements in one row or column of the linear coefficient matrix.	
• <code>mosek.Task.putaveclist</code>	228
Replaces all elements in one or more rows or columns in the linear constraint matrix by new values.	
• <code>mosek.Task.putbound</code>	228
Changes the bound for either one constraint or one variable.	

- `mosek.Task.putboundlist`..... 229
Changes the bounds of constraints or variables.
- `mosek.Task.putboundslice`..... 229
Modifies bounds.
- `mosek.Task.putcfix`..... 230
Replaces the fixed term in the objective.
- `mosek.Task.putcj`..... 230
Modifies one linear coefficient in the objective.
- `mosek.Task.putclist`..... 230
Modifies a part of the linear objective coefficients.
- `mosek.Task.putcone`..... 231
Replaces a conic constraint.
- `mosek.Task.putdoupparam`..... 231
Sets a double parameter.
- `mosek.Task.putintparam`..... 231
Sets an integer parameter.
- `mosek.Task.putmaxnumanz`..... 232
The function changes the size of the preallocated storage for linear coefficients.
- `mosek.Task.putmaxnumcon`..... 232
Sets the number of preallocated constraints in the optimization task.
- `mosek.Task.putmaxnumcone`..... 232
Sets the number of preallocated conic constraints in the optimization task.
- `mosek.Task.putmaxnumqnz`..... 232
Changes the size of the preallocated storage for quadratic terms.
- `mosek.Task.putmaxnumvar`..... 233
Sets the number of preallocated variables in the optimization task.
- `mosek.Task.putnadoupparam`..... 233
Sets a double parameter.
- `mosek.Task.putnaintparam`..... 233
Sets an integer parameter.
- `mosek.Task.putname`..... 233
Assigns a name to a problem item.
- `mosek.Task.putnastrparam`..... 234
Sets a string parameter.
- `mosek.Task.putobjname`..... 234
Assigns a new name to the objective.

• <code>mosek.Task.putobjsense</code>	234
Sets the objective sense.	
• <code>mosek.Task.putparam</code>	234
Modifies the value of parameter.	
• <code>mosek.Task.putqcon</code>	235
Replaces all quadratic terms in constraints.	
• <code>mosek.Task.putqconk</code>	235
Replaces all quadratic terms in a single constraint.	
• <code>mosek.Task.putqobj</code>	236
Replaces all quadratic terms in the objective.	
• <code>mosek.Task.putqobjij</code>	237
Replaces one coefficient in the quadratic term in the objective.	
• <code>mosek.Task.putsolution</code>	237
Inserts a solution.	
• <code>mosek.Task.putsolutioni</code>	238
Sets the primal and dual solution information for a single constraint or variable.	
• <code>mosek.Task.putsolutionyi</code>	239
Inputs the dual variable of a solution.	
• <code>mosek.Task.putstrparam</code>	239
Sets a string parameter.	
• <code>mosek.Task.puttaskname</code>	239
Assigns a new name to the task.	
• <code>mosek.Task.putvarbranchorder</code>	240
Assigns a branching priority and direction to a variable.	
• <code>mosek.Task.putvartype</code>	240
Sets the variable type of one variable.	
• <code>mosek.Task.putvartypelist</code>	240
Sets the variable type for one or more variables.	
• <code>mosek.Task.readbranchpriorities</code>	241
Reads branching priority data from a file.	
• <code>mosek.Task.readdata</code>	241
Reads problem data from a file.	
• <code>mosek.Task.readparamfile</code>	241
Reads a parameter file.	
• <code>mosek.Task.readsolution</code>	241
Reads a solution from a file.	

- `mosek.Task.readsummary` 242
Prints information about last file read.
- `mosek.Task.relaxprimal` 242
Creates a problem that finds the minimal change to the bounds that makes an infeasible problem feasible.
- `mosek.Task.remove` 244
The function removes a number of constraints or variables.
- `mosek.Task.removecone` 244
Removes a conic constraint from the problem.
- `mosek.Task.resizetask` 245
Resizes an optimization task.
- `mosek.Task.sensitivityreport` 245
Creates a sensitivity report.
- `mosek.Task.set_Stream` 245
Attach a stream call-back handler.
- `mosek.Task.setdefaults` 246
Resets all parameters values.
- `mosek.Task.solutiondef` 246
Checks whether a solution is defined.
- `mosek.Task.solutionsummary` 246
Prints a short summary of the current solutions.
- `mosek.Task.solvewithbasis` 246
Solve a linear equation system involving a basis matrix.
- `mosek.Task.strtoconetype` 247
Obtains a cone type code.
- `mosek.Task.strtosk` 247
Obtains a status key.
- `mosek.Task.undefsolution` 248
Undefines a solution.
- `mosek.Task.writebranchpriorities` 248
Writes branching priority data to a file.
- `mosek.Task.writedata` 248
Writes problem data to a file.
- `mosek.Task.writeparamfile` 249
Writes all the parameters to a parameter file.

- `mosek.Task.writesolution`.....249
Write a solution to a file.

- `mosek.Task.analyzeproblem`

Syntax:

```
analyzeproblem (mosek.streamtype whichstream)
               whichstream (input) Index of the stream.
```

Description: The function analyze the data of task and writes out a report.

- `mosek.Task.analyzesolution`

Syntax:

```
analyzesolution (
    mosek.streamtype whichstream,
    mosek.soltype whichsol)
               whichstream (input) Index of the stream.
               whichsol (input) Selects a solution.
```

Description: 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:

- `mosek.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.
- `mosek.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.
- `mosek.dparam.ana_sol_infeas_tol`. The tolerance defining when a constraint is considered violated. If a constraint is violated more than this, it will be listed in the summary.

See also:

`Task.getpeqi` Obtains the primal equation infeasibility.
`Task.getdeqi` Obtains the dual equation infeasibility.
`Task.getpbi` Obtains the primal bound infeasibility.
`Task.getdbi` Obtains the dual bound infeasibility.
`Task.getdcni` Obtains the dual cone infeasibility.
`Task.getpcni` Obtains the primal cone infeasibility.
`Task.getsolutioninf` Obtains information about a solution.
`Task.getsolutionstatus` Obtains information about the problem and solution statuses.

- `mosek.Task.append`

Syntax:

```
append (
    mosek.accmode accmode,
    int num)
```

accmode (input) Defines if operations are performed row-wise (constraint-oriented) or column-wise (variable-oriented).

num (input) Number of constraints or variables which should be appended.

Description: Appends a number of constraints or variables to the model. Appended constraints will be declared free and appended variables will be fixed at the level zero. Please note that MOSEK will automatically expand the problem dimension to accommodate the additional constraints and variables.

See also:

Task.remove The function removes a number of constraints or variables.

- `mosek.Task.appendcone`

Syntax:

```
appendcone (
    mosek.conetype conetype,
    double coneapar,
    array(int) submem)
```

conetype (input) Specifies the type of the cone.

coneapar (input) This argument is currently not used. Can be set to 0.0.

submem (input) Variable subscripts of the members in the cone.

Description: Appends a new conic constraint to the problem. Hence, add a constraint

$$\bar{x} \in \mathcal{C}$$

to the problem where \mathcal{C} is a convex cone. \bar{x} is a subset of the variables which will be specified by the argument **submem**.

Depending on the value of **conetype** this function appends a normal (**mosek.conetype.quad**) or rotated quadratic cone (**mosek.conetype.rquad**). Define

$$\bar{x} = x_{\text{submem}[0]}, \dots, x_{\text{submem}[\text{nummem}-1]}$$

. Depending on the value of **conetype** this function appends one of the constraints:

- Quadratic cone (**mosek.conetype.quad**) :

$$\bar{x}_0 \geq \sqrt{\sum_{i=1}^{i < \text{nummem}} \bar{x}_i^2}$$

- Rotated quadratic cone (**mosek.conetype.rquad**) :

$$2\bar{x}_0\bar{x}_1 \geq \sum_{i=2}^{i < \text{nummem}} \bar{x}_i^2, \quad \bar{x}_0, \bar{x}_1 \geq 0$$

Please note that the sets of variables appearing in different conic constraints must be disjoint.
For an explained code example see Section 5.4.

- `mosek.Task.basiscond`

Syntax:

```
nrmbasis, nrminvbasis = basiscond ()
```

`nrmbasis` (**output**) An estimate for the 1 norm of the basis.

`nrminvbasis` (**output**) An estimate for the 1 norm of the inverse of the basis.

Description: If a basic solution is available and it defines a nonsingular basis, then this function computes the 1-norm estimate of the basis matrix and an 1-norm estimate for the inverse of the basis matrix. The 1-norm estimates are computed using the method outlined in [21, pp. 388-391].

By definition the 1-norm condition number of a matrix B is defined as

$$\kappa_1(B) := \|B\|_1 \|B^{-1}\|_1.$$

Moreover, the larger the condition number is the harder it is to solve linear equation systems involving B . Given estimates for $\|B\|_1$ and $\|B^{-1}\|_1$ it is also possible to estimate $\kappa_1(B)$.

- `mosek.Task.checkconvexity`

Syntax:

```
checkconvexity ()
```

Description: This function checks if a quadratic optimization problem is convex. The amount of checking is controlled by `mosek.iparam.check.convexity`.

The function throws an exception if the problem is not convex.

See also:

`mosek.iparam.check.convexity`

- `mosek.Task.checkdata`

Syntax:

```
checkdata ()
```

Description: Checks the data of the optimization task.

- `mosek.Task.checkmem`

Syntax:

```
checkmem (
    str file,
    int line)
```

`file` (**input**) File from which the function is called.

`line` (**input**) Line in the file from which the function is called.

Description: Checks the memory allocated by the task.

- `mosek.Task.chgbound`

Syntax:

```
chgbound (
    mosek.accmode accmode,
    int i,
    int lower,
    int finite,
    double value)
```

accmode (input) Defines if operations are performed row-wise (constraint-oriented) or column-wise (variable-oriented).

i (input) Index of the constraint or variable for which the bounds should be changed.

lower (input) If non-zero, then the lower bound is changed, otherwise the upper bound is changed.

finite (input) If non-zero, then **value** is assumed to be finite.

value (input) New value for the bound.

Description: Changes a bound for one constraint or variable. If **accmode** equals `mosek.accmode.con`, a constraint bound is changed, otherwise a variable bound is changed.

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 bound, in particular, if the lower and upper bounds are identical, the bound key is changed to **fixed**.

See also:

`Task.putbound` Changes the bound for either one constraint or one variable.

`mosek.dparam.data_tol_bound_inf`

`mosek.dparam.data_tol_bound_wrn`

- `mosek.Task.commitchanges`

Syntax:

```
commitchanges ()
```

Description: Commits all cached problem changes to the task. It is usually not necessary explicitly to call this function since changes will be committed automatically when required.

- `mosek.Task.deletesolution`

Syntax:

```
deletesolution (mosek.soltype whichsol)
whichsol (input) Selects a solution.
```

Description: Undefines a solution and frees the memory it uses.

- `mosek.Task.dualsensitivity`

Syntax:

```
dualsensitivity (
    array(int) subj,
    array(double) leftpricej,
    array(double) rightpricej,
    array(double) leftrangej,
    array(double) rightrangej)

subj (input) Index of objective coefficients to analyze.
leftpricej (output) leftpricej[j] is the left shadow price for the coefficients with index
    subj[j].
rightpricej (output) rightpricej[j] is the right shadow price for the coefficients with
    index subj[j].
leftrangej (output) leftrangej[j] is the left range  $\beta_1$  for the coefficient with index
    subj[j].
rightrangej (output) rightrangej[j] is the right range  $\beta_2$  for the coefficient with index
    subj[j].
```

Description: Calculates sensitivity information for objective coefficients. The indexes of the coefficients to analyze are

$$\{\text{subj}[i] | i \in 0, \dots, \text{numj} - 1\}$$

The results are returned so that e.g `leftprice[j]` is the left shadow price of the objective coefficient with index `subj[j]`.

The type of sensitivity analysis to perform (basis or optimal partition) is controlled by the parameter `mosek.iparam.sensitivity_type`.

For an example, please see Section 12.5.

See also:

`Task.primalsensitivity` Perform sensitivity analysis on bounds.
`Task.sensitivityreport` Creates a sensitivity report.
`mosek.iparam.sensitivity_type`
`mosek.iparam.log_sensitivity`
`mosek.iparam.log_sensitivity_opt`

- `mosek.Task.getaij`

Syntax:

```

    aij = getaij (
        int i,
        int j)

```

i (input) Row index of the coefficient to be returned.

j (input) Column index of the coefficient to be returned.

aij (output) The required coefficient $a_{i,j}$.

Description: Obtains a single coefficient in A .

- `mosek.Task.getapieceenumnz`

Syntax:

```

    numnz = getapieceenumnz (
        int firsti,
        int lasti,
        int firstj,
        int lastj)

```

firsti (input) Index of the first row in the rectangular piece.

lasti (input) Index of the last row plus one in the rectangular piece.

firstj (input) Index of the first column in the rectangular piece.

lastj (input) Index of the last column plus one in the rectangular piece.

numnz (output) Number of non-zero A elements in the rectangular piece.

Description: Obtains the number non-zeros in a rectangular piece of A , i.e. the number

$$|\{(i,j) : a_{i,j} \neq 0, \text{firsti} \leq i \leq \text{lasti} - 1, \text{firstj} \leq j \leq \text{lastj} - 1\}|$$

where $|\mathcal{I}|$ means the number of elements in the set \mathcal{I} .

This function is not an efficient way to obtain the number of non-zeros in one row or column.

In that case use the function `Task.getavecnumnz`.

See also:

`Task.getavecnumnz` Obtains the number of non-zero elements in one row or column of the linear constraint matrix

- `mosek.Task.getaslice`

Syntax:

```

    surp = getaslice (
        mosek.accmode accmode,
        int first,
        int last,
        long surp,
        array(long) ptrb,
        array(long) ptre,
        array(int) sub,
        array(double) val)

```

accmode (input) Defines whether a column slice or a row slice is requested.

first (input) Index of the first row or column in the sequence.

last (input) Index of the last row or column in the sequence **plus one**.

surp (input/output) The required rows and columns are stored sequentially in **sub** and **val** starting from position **maxnumnz-surp[0]**. Upon return **surp** has been decremented by the total number of non-zero elements in the rows and columns obtained.

ptrb (output) **ptrb[t]** is an index pointing to the first element in the *t*th row or column obtained.

ptre (output) **ptre[t]** is an index pointing to the last element plus one in the *t*th row or column obtained.

sub (output) Contains the row or column subscripts.

val (output) Contains the coefficient values.

Description: Obtains a sequence of rows or columns from *A* in sparse format.

See also:

Task.getaslicenumnz64 Obtains the number of non-zeros in a slice of rows or columns of the coefficient matrix.

- `mosek.Task.getaslicenumnz`

Syntax:

```
numnz = getaslicenumnz (
    mosek.accmode accmode,
    int first,
    int last)
```

accmode (input) Defines whether non-zeros are counted in a column slice or a row slice.

first (input) Index of the first row or column in the sequence.

last (input) Index of the last row or column **plus one** in the sequence.

numnz (output) Number of non-zeros in the slice.

Description: Obtains the number of non-zeros in a slice of rows or columns of *A*.

- `mosek.Task.getaslicetrip`

Syntax:

```
surp = getaslicetrip (
    mosek.accmode accmode,
    int first,
    int last,
    int surp,
    array(int) subi,
    array(int) subj,
    array(double) val)
```

accmode (input) Defines whether a column-slice or a row-slice is requested.

first (input) Index of the first row or column in the sequence.

last (input) Index of the last row or column in the sequence **plus one**.

surp (input/output) The required rows and columns are stored sequentially in **subi** and **val** starting from position **maxnumnz-surp[0]**. On return **surp** has been decremented by the total number of non-zero elements in the rows and columns obtained.

subi (output) Constraint subscripts.

subj (output) Variable subscripts.

val (output) Values.

Description: Obtains a sequence of rows or columns from A in a sparse triplet format.

- `mosek.Task.getavec`

Syntax:

```
nzi = getavec (
    mosek.accmode accmode,
    int i,
    array(int) subi,
    array(double) vali)
```

accmode (input) Defines if operations are performed row-wise (constraint-oriented) or column-wise (variable-oriented).

i (input) Index of the row or column.

subi (output) Index of the non-zeros in the vector obtained.

vali (output) Numerical values of the vector to be obtained.

nzi (output) Number of non-zeros in the vector obtained.

Description: Obtains one row or column of A in a sparse format. If **accmode** equals `mosek.accmode.con` a row is returned and hence:

$$\text{vali}[k] = a_{i, \text{subi}[k]}, \quad k = 0, \dots, \text{nzi}[0] - 1$$

If **accmode** equals `mosek.accmode.var` a column is returned, that is:

$$\text{vali}[k] = a_{\text{subi}[k], i}, \quad k = 0, \dots, \text{nzi}[0] - 1.$$

- `mosek.Task.getavecnumnz`

Syntax:

```
nzj = getavecnumnz (
    mosek.accmode accmode,
    int i)
```

accmode (input) Defines whether non-zeros are counted by columns or by rows.

i (input) Index of the row or column.

nzj (output) Number of non-zeros in the i th row or column of A .

Description: Obtains the number of non-zero elements in one row or column of A .

- `mosek.Task.getbound`

Syntax:

```
bk,bl,bu = getbound (
    mosek.accmode accmode,
    int i)
```

accmode (input) Defines if operations are performed row-wise (constraint-oriented) or column-wise (variable-oriented).

i (input) Index of the constraint or variable for which the bound information should be obtained.

bk (output) Bound keys.

bl (output) Values for lower bounds.

bu (output) Values for upper bounds.

Description: Obtains bound information for one constraint or variable.

- `mosek.Task.getboundslice`

Syntax:

```
getboundslice (
    mosek.accmode accmode,
    int first,
    int last,
    array(boundkey) bk,
    array(double) bl,
    array(double) bu)
```

accmode (input) Defines if operations are performed row-wise (constraint-oriented) or column-wise (variable-oriented).

first (input) First index in the sequence.

last (input) Last index plus 1 in the sequence.

bk (output) Bound keys.

bl (output) Values for lower bounds.

bu (output) Values for upper bounds.

Description: Obtains bounds information for a sequence of variables or constraints.

- `mosek.Task.getc`

Syntax:

```
getc (array(double) c)
```

c (output) Linear terms of the objective as a dense vector. The length is the number of variables.

Description: Obtains all objective coefficients *c*.

- `mosek.Task.getcfix`

Syntax:

```
cfix = getcfix ()
```

cfix (output) Fixed term in the objective.

Description: Obtains the fixed term in the objective.

- `mosek.Task.getcone`

Syntax:

```
conetype, coneapar, nummem = getcone (
    int k,
    array(int) submem)
```

k (input) Index of the cone constraint.

submem (output) Variable subscripts of the members in the cone.

conetype (output) Specifies the type of the cone.

coneapar (output) This argument is currently not used. Can be set to 0.0.

nummem (output) Number of member variables in the cone.

Description: Obtains a conic constraint.

- `mosek.Task.getconeinfo`

Syntax:

```
conetype, coneapar, nummem = getconeinfo (int k)
```

k (input) Index of the conic constraint.

conetype (output) Specifies the type of the cone.

coneapar (output) This argument is currently not used. Can be set to 0.0.

nummem (output) Number of member variables in the cone.

Description: Obtains information about a conic constraint.

- `mosek.Task.getconname`

Syntax:

```
name = getconname (int i)
```

i (input) Index.

name (output) Is assigned the required name.

Description: Obtains a name of a constraint.

- `mosek.Task.getcslice`

Syntax:

```
getcslice (
    int first,
    int last,
    array(double) c)
```

first (input) First index in the sequence.

last (input) Last index plus 1 in the sequence.

c (output) Linear terms of the objective as a dense vector. The length is the number of variables.

Description: Obtains a sequence of elements in *c*.

- `mosek.Task.getdbi`

Syntax:

```
getdbi (
    mosek.soltype whichsol,
    mosek.accmode accmode,
    array(int) sub,
    array(double) dbi)
```

whichsol (input) Selects a solution.

accmode (input) If set to `mosek.accmode.con` then `sub` contains constraint indexes, otherwise variable indexes.

sub (input) Indexes of constraints or variables.

dbi (output) Dual bound infeasibility. If `accmode` is `mosek.accmode.con` then

$$\text{dbi}[i] = \max(-(s_l^c)_{\text{sub}[i]}, -(s_u^c)_{\text{sub}[i]}, 0) \quad \text{for } i = 0, \dots, \text{len} - 1 \quad (14.1)$$

else

$$\text{dbi}[i] = \max(-(s_l^x)_{\text{sub}[i]}, -(s_u^x)_{\text{sub}[i]}, 0) \quad \text{for } i = 0, \dots, \text{len} - 1. \quad (14.2)$$

Description: Obtains the dual bound infeasibility.

See also:

`Task.getsolutioninf` Obtains information about a solution.

- `mosek.Task.getdcni`

Syntax:

```
getdcni (
    mosek.soltype whichsol,
    array(int) sub,
    array(double) dcni)
```

whichsol (input) Selects a solution.

sub (input) Constraint indexes to calculate equation infeasibility for.

dcni (output) `dcni[i]` contains dual cone infeasibility for the cone with index `sub[i]`.

Description: Obtains the dual cone infeasibility.

See also:

`Task.getsolutioninf` Obtains information about a solution.

- `mosek.Task.getdeqi`

Syntax:

```
getdeqi (
    mosek.soltype whichsol,
    mosek.accmode accmode,
    array(int) sub,
    array(double) deqi,
    int normalize)
```

whichsol (input) Selects a solution.

accmode (input) If set to `mosek.accmode.con` the dual equation infeasibilitys corresponding to constraints are retrieved. Otherwise for a variables.

sub (input) Indexes of constraints or variables.

deqi (output) Dual equation infeasibilitys corresponding to constraints or variables.

normalize (input) If non-zero, normalize with largest absolute value of the input data used to compute the individual infeasibility.

Description: Obtains the dual equation infeasibility. If `acmode` is `mosek.accmode.con` then

$$\text{pbi}[i] = |(-y + s_l^c - s_u^c)_{\text{sub}[i]}| \quad \text{for } i = 0, \dots, \text{len} - 1 \quad (14.3)$$

If `acmode` is `mosek.accmode.var` then

$$\text{pbi}[i] = |(A^T y + s_l^x - s_u^x - c)_{\text{sub}[i]}| \quad \text{for } i = 0, \dots, \text{len} - 1 \quad (14.4)$$

See also:

`Task.getsolutioninf` Obtains information about a solution.

- `mosek.Task.getdouinf`

Syntax:

```
dvalue = getdouinf (mosek.dinfitem whichdinf)
```

whichdinf (input) A double information item. See section 17.13 for the possible values.

dvalue (output) The value of the required double information item.

Description: Obtains a double information item from the task information database.

- `mosek.Task.getdoupam`

Syntax:

```
parvalue = getdoupam (mosek.dparam param)
```

param (input) Which parameter.

parvalue (output) Parameter value.

Description: Obtains the value of a double parameter.

- `mosek.Task.getdualobj`

Syntax:

```
dualobj = getdualobj (mosek.soltype whichsol)
whichsol (input) Selects a solution.
dualobj (output) Objective value corresponding to the dual solution.
```

Description: Obtains the current objective value of the dual problem for **whichsol**.

- `mosek.Task.getinfeasiblesubproblem`

Syntax:

```
inftask = getinfeasiblesubproblem (mosek.soltype whichsol)
whichsol (input) Which solution to use when determining the infeasible subproblem.
inftask (output) A new task containing the infeasible subproblem.
```

Description: Obtains an infeasible subproblem. The infeasible subproblem is a problem consisting of a subset of the original constraints such that the problem is still infeasible. For more information see Section 10.2.

See also:

`mosek.iparam.infeas_prefer_primal`
Task.relaxprimal Creates a problem that finds the minimal change to the bounds that makes an infeasible problem feasible.

- `mosek.Task.getinti`

Syntax:

```
getinti (
    mosek.soltype whichsol,
    array(int) sub,
    array(double) inti)
whichsol (input) Selects a solution.
sub (input) Variable indexes for which to calculate the integer infeasibility.
inti (output) inti[i] contains integer infeasibility of variable sub[i].
```

Description: Obtains the primal equation infeasibility.

$$\text{peqi}[i] = |(Ax - x^c)_{\text{sub}[i]}| \quad \text{for } i = 0, \dots, \text{len} - 1. \quad (14.5)$$

See also:

Task.getsolutioninf Obtains information about a solution.

- `mosek.Task.getintinf`

Syntax:

```
ivalue = getintinf (mosek.iinfitem whichiinf)
whichiinf (input) Specifies an information item.
```

ivalue (**output**) The value of the required integer information item.

Description: Obtains an integer information item from the task information database.

- `mosek.Task.getintparam`

Syntax:

```
parvalue = getintparam (mosek.iparam param)
```

`param` (**input**) Which parameter.

`parvalue` (**output**) Parameter value.

Description: Obtains the value of an integer parameter.

- `mosek.Task.getintpntnumthreads`

Syntax:

```
numthreads = getintpntnumthreads ()
```

`numthreads` (**output**) The number of threads.

Description: Obtains the number of threads used by the interior-point optimizer. If `mosek.iparam.intpnt_num_threads` is set to zero this function will return the number of cores on the system. Otherwise it return the value of `mosek.iparam.intpnt_num_threads`.

- `mosek.Task.getlintinf`

Syntax:

```
ivalue = getlintinf (mosek.liinfitem whichliinf)
```

`whichliinf` (**input**) Specifies an information item.

`ivalue` (**output**) The value of the required integer information item.

Description: Obtains an integer information item from the task information database.

- `mosek.Task.getmaxnumanz`

Syntax:

```
maxnumanz = getmaxnumanz ()
```

`maxnumanz` (**output**) Number of preallocated non-zero elements in A .

Description: Obtains number of preallocated non-zeros in A . When this number of non-zeros is reached MOSEK will automatically allocate more space for A .

- `mosek.Task.getmaxnumcon`

Syntax:

```
maxnumcon = getmaxnumcon ()
```

`maxnumcon` (**output**) Number of preallocated constraints in the optimization task.

Description: 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.

- `mosek.Task.getmaxnumcone`

Syntax:

```
maxnumcone = getmaxnumcone ()
```

`maxnumcone` (**output**) Number of preallocated conic constraints in the optimization task.

Description: 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.

- `mosek.Task.getmaxnumqnz`

Syntax:

```
maxnumqnz = getmaxnumqnz ()
```

`maxnumqnz` (**output**) Number of non-zero elements preallocated in quadratic coefficient matrices.

Description: 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 .

- `mosek.Task.getmaxnumvar`

Syntax:

```
maxnumvar = getmaxnumvar ()
```

`maxnumvar` (**output**) Number of preallocated variables in the optimization task.

Description: Obtains the number of preallocated variables in the optimization task. When this number of variables is reached MOSEK will automatically allocate more space for constraints.

- `mosek.Task.getmemusage`

Syntax:

```
meminuse,maxmemuse = getmemusage ()
```

`meminuse` (**output**) Amount of memory currently used by the `task`.

`maxmemuse` (**output**) Maximum amount of memory used by the `task` until now.

Description: Obtains information about the amount of memory used by a task.

- `mosek.Task.getname`

Syntax:

```
len,name = getname (
    mosek.problemitem whichitem,
    int i)
```

`whichitem` (**input**) Problem item, i.e. a cone, a variable or a constraint name..

`i` (**input**) Index.

len (output) Is assigned the length of the required name.

name (output) Is assigned the required name.

Description: Obtains a name of a problem item, i.e. a cone, a variable or a constraint.

- `mosek.Task.getname`

Syntax:

```
name = getname (
    mosek.problemitem whichitem,
    int i)
```

whichitem (input) Problem item, i.e. a cone, a variable or a constraint name..

i (input) Index.

name (output) Is assigned the required name.

Description: Obtains a name of a problem item, i.e. a cone, a variable or a constraint.

- `mosek.Task.getnameindex`

Syntax:

```
asgn,index = getnameindex (
    mosek.problemitem whichitem,
    str name)
```

whichitem (input) Problem item, i.e. a cone, a variable or a constraint name..

name (input) The name which should be checked.

asgn (output) Is non-zero if **name** is assigned.

index (output) If the **name** identifies an item in the task, then **index** is assigned the index of that item.

Description: Checks if a given name identifies a cone, a constraint or a variable in the **task**.

If it does, the index of that item is assigned to **index**, and a non-zero value is assigned to **asgn**. If the name does not identify a problem item, **asgn** is assigned a zero.

- `mosek.Task.getnamelen`

Syntax:

```
len = getnamelen (
    mosek.problemitem whichitem,
    int i)
```

whichitem (input) Problem item, i.e. a cone, a variable or a constraint name..

i (input) Index.

len (output) Is assigned the length of the required name.

Description: Obtains the length of a problem item name.

See also:

Task.getname64 Obtains the name of a cone, a variable or a constraint.

- `mosek.Task.getnumanz`

Syntax:

```
numanz = getnumanz ()
```

`numanz (output)` Number of non-zero elements in A .

Description: Obtains the number of non-zeros in A .

- `mosek.Task.getnumanz64`

Syntax:

```
numanz = getnumanz64 ()
```

`numanz (output)` Number of non-zero elements in A .

Description: Obtains the number of non-zeros in A .

- `mosek.Task.getnumcon`

Syntax:

```
numcon = getnumcon ()
```

`numcon (output)` Number of constraints.

Description: Obtains the number of constraints.

- `mosek.Task.getnumcone`

Syntax:

```
numcone = getnumcone ()
```

`numcone (output)` Number conic constraints.

Description: Obtains the number of cones.

- `mosek.Task.getnumconemem`

Syntax:

```
nummem = getnumconemem (int k)
```

`k (input)` Index of the cone.

`nummem (output)` Number of member variables in the cone.

Description: Obtains the number of members in a cone.

- `mosek.Task.getnumintvar`

Syntax:

```
numintvar = getnumintvar ()
```

`numintvar (output)` Number of integer variables.

Description: Obtains the number of integer-constrained variables.

- `mosek.Task.getnumparam`

Syntax:

`numparam = getnumparam (mosek.parametertype partype)`

partype (input) Parameter type.

numparam (output) Identical to the number of parameters of the type **partype**.

Description: Obtains the number of parameters of a given type.

- `mosek.Task.getnumqconknz`

Syntax:

`numqcnz = getnumqconknz (int k)`

k (input) Index of the constraint for which the number of non-zero quadratic terms should be obtained.

numqcnz (output) Number of quadratic terms. See (5.32).

Description: Obtains the number of non-zero quadratic terms in a constraint.

- `mosek.Task.getnumqconknz64`

Syntax:

`numqcnz = getnumqconknz64 (int k)`

k (input) Index of the constraint for which the number quadratic terms should be obtained.

numqcnz (output) Number of quadratic terms. See (5.32).

Description: Obtains the number of non-zero quadratic terms in a constraint.

- `mosek.Task.getnumqobjnz`

Syntax:

`numqonz = getnumqobjnz ()`

numqonz (output) Number of non-zero elements in Q^o .

Description: Obtains the number of non-zero quadratic terms in the objective.

- `mosek.Task.getnumqobjnz64`

Syntax:

`numqonz = getnumqobjnz64 ()`

numqonz (output) Number of non-zero elements in Q^o .

Description: Obtains the number of non-zero quadratic terms in the objective.

- `mosek.Task.getnumvar`

Syntax:

`numvar = getnumvar ()`

`numvar (output)` Number of variables.

Description: Obtains the number of variables.

- `mosek.Task.getobjname`

Syntax:

```
len,objname = getobjname ()
```

`len (output)` Assigned the length of the objective name.

`objname (output)` Assigned the objective name.

Description: Obtains the name assigned to the objective function.

- `mosek.Task.getobjsense`

Syntax:

```
sense = getobjsense ()
```

`sense (output)` The returned objective sense.

Description: Gets the objective sense of the task.

See also:

`Task.putobjsense` Sets the objective sense.

- `mosek.Task.getpbi`

Syntax:

```
getpbi (
    mosek.soltype whichsol,
    mosek.accmode accmode,
    array(int) sub,
    array(double) pbi,
    int normalize)
```

`whichsol (input)` Selects a solution.

`accmode (input)` If set to `mosek.accmode.var` return bound infeasibility for x otherwise for x^c .

`sub (input)` An array of constraint or variable indexes.

`pbi (output)` Bound infeasibility for x or x^c .

`normalize (input)` If non-zero, normalize with largest absolute value of the input data used to compute the individual infeasibility.

Description: Obtains the primal bound infeasibility. If `acmode` is `mosek.accmode.con` then

$$pbi[i] = \max(x_{sub[i]}^c - u_{sub[i]}^c, l_{sub[i]}^c - x_{sub[i]}^c, 0) \quad \text{for } i = 0, \dots, len - 1 \quad (14.6)$$

If `acmode` is `mosek.accmode.var` then

$$pbi[i] = \max(x_{sub[i]} - u_{sub[i]}^x, l_{sub[i]}^x - x_{sub[i]}, 0) \quad \text{for } i = 0, \dots, len - 1 \quad (14.7)$$

See also:

`Task.getsolutioninf` Obtains information about a solution.

- `mosek.Task.getpcni`

Syntax:

```
getpcni (
    mosek.soltype whichsol,
    array(int) sub,
    array(double) pcni)
```

`whichsol` (**input**) Selects a solution.

`sub` (**input**) Constraint indexes for which to calculate the equation infeasibility.

`pcni` (**output**) `pcni[i]` contains primal cone infeasibility for the cone with index `sub[i]`.

Description: Obtains the primal cone infeasibility.

See also:

`Task.getsolutioninf` Obtains information about a solution.

- `mosek.Task.getpeqi`

Syntax:

```
getpeqi (
    mosek.soltype whichsol,
    array(int) sub,
    array(double) peqi,
    int normalize)
```

`whichsol` (**input**) Selects a solution.

`sub` (**input**) Constraint indexes for which to calculate the equation infeasibility.

`peqi` (**output**) `peqi[i]` contains equation infeasibility of constraint `sub[i]`.

`normalize` (**input**) If non-zero, normalize with largest absolute value of the input data used to compute the individual infeasibility.

Description: Obtains the primal equation infeasibility.

$$\text{peqi}[i] = |(|(Ax - x^c)_{\text{sub}[i]}|) \quad \text{for } i = 0, \dots, \text{len} - 1. \quad (14.8)$$

See also:

`Task.getsolutioninf` Obtains information about a solution.

- `mosek.Task.getprimalobj`

Syntax:

```
primalobj = getprimalobj (mosek.soltype whichsol)
```

`whichsol` (**input**) Selects a solution.

`primalobj` (**output**) Objective value corresponding to the primal solution.

Description: Obtains the primal objective value for a solution.

- `mosek.Task.getproptype`

Syntax:

```
proptype = getproptype ()
```

proptype (output) The problem type.

Description: Obtains the problem type.

- `mosek.Task.getqconk`

Syntax:

```
qcsurp,numqcncz = getqconk (
    int k,
    int qcsurp,
    array(int) qcsubi,
    array(int) qcsubj,
    array(double) qcval)
```

k (input) Which constraint.

qcsurp (input/output) When entering the function it is assumed that the last `qcsurp[0]` positions in `qcsubi`, `qcsubj`, and `qcval` are free. Hence, the quadratic terms are stored in this area, and upon return `qcsurp` is number of free positions left in `qcsubi`, `qcsubj`, and `qcval`.

qcsubi (output) i subscripts for q_{ij}^k . See (5.32).

qcsubj (output) j subscripts for q_{ij}^k . See (5.32).

qcval (output) Numerical value for q_{ij}^k .

numqcncz (output) Number of quadratic terms. See (5.32).

Description: Obtains all the quadratic terms in a constraint. The quadratic terms are stored sequentially `qcsubi`, `qcsubj`, and `qcval`.

- `mosek.Task.getqconk64`

Syntax:

```
qcsurp,numqcncz = getqconk64 (
    int k,
    long qcsurp,
    array(int) qcsubi,
    array(int) qcsubj,
    array(double) qcval)
```

k (input) Which constraint.

qcsurp (input/output) When entering the function it is assumed that the last `qcsurp[0]` positions in `qcsubi`, `qcsubj`, and `qcval` are free. Hence, the quadratic terms are stored in this area, and upon return `qcsurp` is number of free positions left in `qcsubi`, `qcsubj`, and `qcval`.

qcsubi (output) i subscripts for q_{ij}^k . See (5.32).
qcsubj (output) j subscripts for q_{ij}^k . See (5.32).
qcval (output) Numerical value for q_{ij}^k .
numqcnz (output) Number of quadratic terms. See (5.32).

Description: Obtains all the quadratic terms in a constraint. The quadratic terms are stored sequentially **qcsubi**, **qcsubj**, and **qcval**.

- `mosek.Task.getqobj`

Syntax:

```

qosurp,numqonz = getqobj (
    int qosurp,
    array(int) qosubi,
    array(int) qosubj,
    array(double) qoval)

```

qosurp (input/output) When entering the function **qosurp**[0] is the number of free positions at the end of the arrays **qosubi**, **qosubj**, and **qoval**, and upon return **qosurp** is the updated number of free positions left in those arrays.

qosubi (output) i subscript for q_{ij}^o .
qosubj (output) j subscript for q_{ij}^o .
qoval (output) Numerical value for q_{ij}^o .
numqonz (output) Number of non-zero elements in Q^o .

Description: Obtains the quadratic terms in the objective. The required quadratic terms are stored sequentially in **qosubi**, **qosubj**, and **qoval**.

- `mosek.Task.getqobj64`

Syntax:

```

qosurp,numqonz = getqobj64 (
    long qosurp,
    array(int) qosubi,
    array(int) qosubj,
    array(double) qoval)

```

qosurp (input/output) When entering the function **qosurp**[0] is the number of free positions at the end of the arrays **qosubi**, **qosubj**, and **qoval**, and upon return **qosurp** is the updated number of free positions left in those arrays.

qosubi (output) i subscript for q_{ij}^o .
qosubj (output) j subscript for q_{ij}^o .
qoval (output) Numerical value for q_{ij}^o .
numqonz (output) Number of non-zero elements in Q^o .

Description: Obtains the quadratic terms in the objective. The required quadratic terms are stored sequentially in **qosubi**, **qosubj**, and **qoval**.

- `mosek.Task.getqobjij`

Syntax:

```
qobj = getqobjij (
    int i,
    int j)
i (input) Row index of the coefficient.
j (input) Column index of coefficient.
qobj (output) The required coefficient.
```

Description: Obtains one coefficient q_{ij}^o in the quadratic term of the objective.

- `mosek.Task.getreducedcosts`

Syntax:

```
getreducedcosts (
    mosek.soltype whichsol,
    int first,
    int last,
    array(double) redcosts)
whichsol (input) Selects a solution.
first (input) See formula (14.9) for the definition.
last (input) See formula (14.9) for the definition.
redcosts (output) The reduced costs in the required sequence of variables are stored
    sequentially in redcosts starting at redcosts[0].
```

Description: Computes the reduced costs for a sequence of variables and return them in the variable **redcosts** i.e.

$$\text{redcosts}[j - \text{first}] = (s_l^x)_j - (s_u^x)_j, \quad j = \text{first}, \dots, \text{last} - 1. \quad (14.9)$$

- `mosek.Task.getsolution`

Syntax:

```
prosta,solsta = getsolution (
    mosek.soltype whichsol,
    array(stakey) skc,
    array(stakey) skx,
    array(stakey) skn,
    array(double) xc,
    array(double) xx,
    array(double) y,
    array(double) slc,
    array(double) suc,
    array(double) slx,
    array(double) sux,
    array(double) snx)
```

whichsol (input) Selects a solution.
skc (output) Status keys for the constraints.
skx (output) Status keys for the variables.
skn (output) Status keys for the conic constraints.
xc (output) Primal constraint solution.
xx (output) Primal variable solution (x).
y (output) Vector of dual variables corresponding to the constraints.
slc (output) Dual variables corresponding to the lower bounds on the constraints (s_l^c).
suc (output) Dual variables corresponding to the upper bounds on the constraints (s_u^c).
slx (output) Dual variables corresponding to the lower bounds on the variables (s_l^x).
sux (output) Dual variables corresponding to the upper bounds on the variables (appears as s_u^x).
snx (output) Dual variables corresponding to the conic constraints on the variables (s_n^x).
prosta (output) Problem status.
solsta (output) Solution status.

Description: Obtains the complete solution.

Consider the case of linear programming. The primal problem is given by

$$\begin{aligned}
 &\text{minimize} && c^T x + c^f \\
 &\text{subject to} && \begin{array}{ccc} l^c & \leq & Ax & \leq & u^c, \\ l^x & \leq & x & \leq & u^x. \end{array}
 \end{aligned} \tag{14.10}$$

and the corresponding dual problem is

$$\begin{aligned}
 &\text{maximize} && (l^c)^T s_l^c - (u^c)^T s_u^c \\
 &&& + (l^x)^T s_l^x - (u^x)^T s_u^x + c^f \\
 &\text{subject to} && \begin{array}{ccc} A^T y + s_l^x - s_u^x & = & c, \\ -y + s_l^c - s_u^c & = & 0, \\ s_l^c, s_u^c, s_l^x, s_u^x & \geq & 0. \end{array}
 \end{aligned} \tag{14.11}$$

In this case the mapping between variables and arguments to the function is as follows:

xx: Corresponds to variable x .
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 .
xc: Corresponds to Ax .

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:

mosek.solsta.optimal An optimal solution satisfying the optimality criteria for continuous problems is returned.

`mosek.solsta.integer_optimal` An optimal solution satisfying the optimality criteria for integer problems is returned.

`mosek.solsta.prim_feas` A solution satisfying the feasibility criteria.

`mosek.solsta.prim_infeas_cer` A primal certificate of infeasibility is returned.

`mosek.solsta.dual_infeas_cer` A dual certificate of infeasibility is returned.

See also:

`Task.getsolutioni` Obtains the solution for a single constraint or variable.

`Task.getsolutionslice` Obtains a slice of the solution.

- `mosek.Task.getsolutioni`

Syntax:

```
sk,x,sl,su,sn = getsolutioni (
    mosek.accmode accmode,
    int i,
    mosek.soltype whichsol)
```

accmode (input) If set to `mosek.accmode.con` the solution information for a constraint is retrieved. Otherwise for a variable.

i (input) Index of the constraint or variable.

whichsol (input) Selects a solution.

sk (output) Status key of the constraint or variable.

x (output) Solution value of the primal variable.

sl (output) Solution value of the dual variable associated with the lower bound.

su (output) Solution value of the dual variable associated with the upper bound.

sn (output) Solution value of the dual variable associated with the cone constraint.

Description: Obtains the primal and dual solution information for a single constraint or variable.

See also:

`Task.getsolution` Obtains the complete solution.

`Task.getsolutionslice` Obtains a slice of the solution.

- `mosek.Task.getsolutioninf`

Syntax:

```
prosta,solsta,primalobj,maxpbi,maxpcni,maxpeqi,maxinti,dualobj,maxdbi,maxdcni,maxdeqi = getsolutioninf
```

whichsol (input) Selects a solution.

prosta (output) Problem status.

solsta (output) Solution status.

primalobj (output) Value of the primal objective.

$$c^T x + c^f \quad (14.12)$$

maxpbi (output) Maximum infeasibility in primal bounds on variables.

$$\max \left\{ 0, \max_{i \in 1, \dots, n-1} (x_i - u_i^x), \max_{i \in 1, \dots, n-1} (l_i^x - x_i), \max_{i \in 1, \dots, n-1} (x_i^c - u_i^c), \max_{i \in 1, \dots, n-1} (l_i^c - x_i^c) \right\} \quad (14.13)$$

maxpcni (output) Maximum infeasibility in the primal conic constraints.

maxpeqi (output) Maximum infeasibility in primal equality constraints.

$$\|Ax - x^c\|_\infty \quad (14.14)$$

maxinti (output) Maximum infeasibility in integer constraints.

$$\max_{i \in \{0, \dots, n-1\}} (\min(x_i - \lfloor x_i \rfloor, \lceil x_i \rceil - x_i)). \quad (14.15)$$

dualobj (output) Value of the dual objective.

$$(l^c)^T s_l^c - (u^c)^T s_u^c + c^f \quad (14.16)$$

maxdbi (output) Maximum infeasibility in bounds on dual variables.

$$\max \{ 0, \max_{i \in \{0, \dots, n-1\}} -(s_l^x)_i, \max_{i \in \{0, \dots, n-1\}} -(s_u^x)_i, \max_{i \in \{0, \dots, m-1\}} -(s_l^c)_i, \max_{i \in \{0, \dots, m-1\}} -(s_u^c)_i \} \quad (14.17)$$

maxdcni (output) Maximum infeasibility in the dual conic constraints.

maxdeqi (output) Maximum infeasibility in the dual equality constraints.

$$\max \{ \|A^T y + s_l^x - s_u^x - c\|_\infty, \|-y + s_l^c - s_u^c\|_\infty \} \quad (14.18)$$

Description: Obtains information about the quality of a solution. Part of the following documentation is restricted to the linear case, it should be clear how to extend to other problem classes.

When optimizing MOSEK solves a reformulated problem with only equality constraints.

$$\begin{aligned} & \text{minimize} && c^T x + c^f \\ & \text{subject to} && Ax - x^c = 0 \\ & && l^x \leq x \leq u^x, \\ & && l^c \leq x^c \leq u^c. \end{aligned} \quad (14.19)$$

where

$$x^c \in \mathbb{R}^m \text{ and } x \in \mathbb{R}^n.$$

and the corresponding dual problem is

$$\begin{aligned} & \text{maximize} && (l^c)^T s_l^c - (u^c)^T s_u^c + c^f \\ & && + (l^x)^T s_l^x - (u^x)^T s_u^x + c^f \\ & \text{subject to} && A^T y + s_l^x - s_u^x = c, \\ & && -y + s_l^c - s_u^c = 0, \\ & && s_l^c, s_u^c, s_l^x, s_u^x \geq 0. \end{aligned} \quad (14.20)$$

The values returned by this function refers to these problems.

Please note that this function computes the objective value and other values every time it is called. Hence, for efficiency reasons this function should not be used too frequently.

If only the problem status or the solution status is required then use `Task.getsolutionstatus` instead.

See also:

`Task.getpeqi` Obtains the primal equation infeasibility.

`Task.getdeqi` Obtains the dual equation infeasibility.

`Task.getpbi` Obtains the primal bound infeasibility.

`Task.getdbi` Obtains the dual bound infeasibility.

`Task.getdcni` Obtains the dual cone infeasibility.

`Task.getpcni` Obtains the primal cone infeasibility.

`Task.analyzesolution` Print information related to the quality of the solution.

`Task.getsolutionstatus` Obtains information about the problem and solution statuses.

- `mosek.Task.getsolutionslice`

Syntax:

```
getsolutionslice (
    mosek.soltype whichsol,
    mosek.solitem solitem,
    int first,
    int last,
    array(double) values)
```

whichsol (input) Selects a solution.

solitem (input) Which part of the solution is required.

first (input) Index of the first value in the slice.

last (input) Value of the last index+1 in the slice, e.g. if `xx[5,...,9]` is required **last** should be 10.

values (output) The values in the required sequence are stored sequentially in **values** starting at **values[0]**.

Description: Obtains a slice of the solution.

Consider the case of linear programming. The primal problem is given by

$$\begin{aligned} & \text{minimize} && c^T x + c^f \\ & \text{subject to} && \begin{array}{lll} l^c & \leq & Ax & \leq & u^c, \\ l^x & \leq & x & \leq & u^x. \end{array} \end{aligned} \quad (14.21)$$

and the corresponding dual problem is

$$\begin{aligned} & \text{maximize} && (l^c)^T s_l^c - (u^c)^T s_u^c \\ & && + (l^x)^T s_l^x - (u^x)^T s_u^x + c^f \\ & \text{subject to} && \begin{array}{lll} A^T y + s_l^x - s_u^x & = & c, \\ -y + s_l^c - s_u^c & = & 0, \\ s_l^c, s_u^c, s_l^x, s_u^x & \geq & 0. \end{array} \end{aligned} \quad (14.22)$$

The `solitem` argument determines which part of the solution is returned:

`mosek.solitem.xx`: The variable `values` return x .
`mosek.solitem.y`: The variable `values` return y .
`mosek.solitem.slc`: The variable `values` return s_l^c .
`mosek.solitem.suc`: The variable `values` return s_u^c .
`mosek.solitem.slx`: The variable `values` return s_l^x .
`mosek.solitem.sux`: The variable `values` return s_u^x .

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:

$$\begin{aligned}
 & \text{maximize} && (l^c)^T s_l^c - (u^c)^T s_u^c \\
 & && + (l^x)^T s_l^x - (u^x)^T s_u^x + c^f \\
 & \text{subject to} && A^T y + s_l^x - s_u^x + s_n^x = c, \\
 & && -y + s_l^c - s_u^c = 0, \\
 & && s_l^c, s_u^c, s_l^x, s_u^x \geq 0, \\
 & && s_n^x \in \mathcal{C}^*
 \end{aligned} \tag{14.23}$$

This introduces one additional dual variable s_n^x . This variable can be accessed by selecting `solitem` as `mosek.solitem.snx`.

The meaning of the values returned by this function also depends on the *solution status* which can be obtained with `Task.getsolutionstatus`. Depending on the solution status value will be:

`mosek.solsta.optimal` A part of the optimal solution satisfying the optimality criteria for continuous problems.
`mosek.solsta.integer_optimal` A part of the optimal solution satisfying the optimality criteria for integer problems.
`mosek.solsta.prim_feas` A part of the solution satisfying the feasibility criteria.
`mosek.solsta.prim_infeas_cer` A part of the primal certificate of infeasibility.
`mosek.solsta.dual_infeas_cer` A part of the dual certificate of infeasibility.

See also:

`Task.getsolution` Obtains the complete solution.

`Task.getsolutioni` Obtains the solution for a single constraint or variable.

- `mosek.Task.getsolutionstatus`

Syntax:

```
prosta,solsta = getsolutionstatus (mosek.soltype whichsol)
```

`whichsol` (input) Selects a solution.

`prosta` (output) Problem status.

`solsta` (output) Solution status.

Description: Obtains information about the problem and solution statuses.

- `mosek.Task.getsolutionstatuskeyslice`

Syntax:

```

getsolutionstatuskeyslice (
    mosek.accmode accmode,
    mosek.soltype whichsol,
    int first,
    int last,
    array(stakey) sk)

```

accmode (input) Defines if operations are performed row-wise (constraint-oriented) or column-wise (variable-oriented).

whichsol (input) Selects a solution.

first (input) Index of the first value in the slice.

last (input) Value of the last index+1 in the slice, e.g. if $xx[5, \dots, 9]$ is required **last** should be 10.

sk (output) The status keys in the required sequence are stored sequentially in **sk** starting at **sk[0]**.

Description: Obtains a slice of the solution status keys.

See also:

Task.getsolution Obtains the complete solution.

Task.getsolutioni Obtains the solution for a single constraint or variable.

- `mosek.Task.gettaskname64`

Syntax:

```
len, taskname = gettaskname64 ()
```

len (output) Is assigned the length of the task name.

taskname (output) Is assigned the task name.

Description: Obtains the name assigned to the task.

- `mosek.Task.getvarbranchdir`

Syntax:

```
direction = getvarbranchdir (int j)
```

j (input) Index of the variable.

direction (output) The branching direction assigned to variable j .

Description: Obtains the branching direction for a given variable j .

- `mosek.Task.getvarbranchpri`

Syntax:

```
priority = getvarbranchpri (int j)
```

j (input) Index of the variable.

priority (output) The branching priority assigned to variable j .

Description: Obtains the branching priority for a given variable j .

- `mosek.Task.getvarname`

Syntax:

```
name = getvarname (int i)
i (input) Index.
name (output) Is assigned the required name.
```

Description: Obtains a name of a variable.

- `mosek.Task.getvartype`

Syntax:

```
vartype = getvartype (int j)
j (input) Index of the variable.
vartype (output) Variable type of variable j.
```

Description: Gets the variable type of one variable.

- `mosek.Task.getvartypelist`

Syntax:

```
getvartypelist (
    array(int) subj,
    array(variabletype) vartype)
subj (input) A list of variable indexes.
vartype (output) The variables types corresponding to the variables specified by subj.
```

Description: Obtains the variable type of one or more variables.

Upon return `vartype[k]` is the variable type of variable `subj[k]`.

- `mosek.Task.initbasissolve`

Syntax:

```
initbasissolve (array(int) basis)
basis (output) 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 .

Description: 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 exception `mosek.rescode.err_basis_singular` is generated.

- `mosek.Task.inputdata`

Syntax:

```
inputdata (
    int maxnumcon,
    int maxnumvar,
    array(double) c,
    double cfix,
    array(long) aptrb,
    array(long) aptre,
    array(int) asub,
    array(double) aval,
    array(boundkey) bkc,
    array(double) blc,
    array(double) buc,
    array(boundkey) bkc,
    array(double) blx,
    array(double) bux)
```

maxnumcon (input) Number of preallocated constraints in the optimization task.

maxnumvar (input) Number of preallocated variables in the optimization task.

c (input) Linear terms of the objective as a dense vector. The length is the number of variables.

cfix (input) Fixed term in the objective.

aptrb (input) Pointer to the first element in the rows or the columns of A . See (5.33) and Section 5.8.3.

aptre (input) Pointers to the last element + 1 in the rows or the columns of A . See (5.33) and Section 5.8.3

asub (input) Coefficient subscripts. See (5.33) and Section 5.8.3.

aval (input) Coefficient values. See (5.33) and Section 5.8.3.

bkc (input) Bound keys for the constraints.

blc (input) Lower bounds for the constraints.

buc (input) Upper bounds for the constraints.

bkc (input) Bound keys for the variables.

blx (input) Lower bounds for the variables.

bux (input) Upper bounds for the variables.

Description: Input the linear part of an optimization problem.

The non-zeros of A are inputted column-wise in the format described in Section 5.8.3.2.

For an explained code example see Section 5.2 and Section 5.8.3.

- `mosek.Task.isdoupname`

Syntax:

```
param = isdoupname (str parname)
```

`parname (input)` Parameter name.
`param (output)` Which parameter.

Description: Checks whether `parname` is a valid double parameter name.

- `mosek.Task.isintparname`

Syntax:

```
param = isintparname (str parname)
parname (input) Parameter name.
param (output) Which parameter.
```

Description: Checks whether `parname` is a valid integer parameter name.

- `mosek.Task.isstrparname`

Syntax:

```
param = isstrparname (str parname)
parname (input) Parameter name.
param (output) Which parameter.
```

Description: Checks whether `parname` is a valid string parameter name.

- `mosek.Task.linkfiletostream`

Syntax:

```
linkfiletostream (
    mosek.streamtype whichstream,
    str filename,
    int append)
```

`whichstream (input)` Index of the stream.

`filename (input)` The name of the file where text from the stream defined by `whichstream` is written.

`append (input)` If this argument is 0 the output file will be overwritten, otherwise text is append to the output file.

Description: Directs all output from a task stream to a file.

- `mosek.Task.makesolutionstatusunknown`

Syntax:

```
makesolutionstatusunknown (mosek.soltype whichsol)
whichsol (input) Selects a solution.
```

Description: Sets the solution status to unknown. Also all the status keys for the constraints and the variables are set to unknown.

- `mosek.Task.netextraction`

Syntax:

```

numcon,numvar = netextraction (
    array(int) netcon,
    array(int) netvar,
    array(double) scalcon,
    array(double) scalvar,
    array(double) cx,
    array(boundkey) bkc,
    array(double) blc,
    array(double) buc,
    array(boundkey) bkx,
    array(double) blx,
    array(double) bux,
    array(int) from,
    array(int) to)

```

netcon (output) Indexes of network constraints (nodes) in the embedded network.

netvar (output) Indexes of network variables (arcs) in the embedded network.

scalcon (output) Scaling values on constraints, used to convert the original part of the problem into network form.

scalvar (output) Scaling values on variables, used to convert the original part of the problem into network form.

cx (output) Linear terms of the objective for variables (arcs) in the embedded network as a dense vector.

bkc (output) Bound keys for the constraints (nodes) in the embedded network.

blc (output) Lower bounds for the constraints (nodes) in the embedded network.

buc (output) Upper bounds for the constraints (nodes) in the embedded network.

bkx (output) Bound keys for the variables (arcs) in the embedded network.

blx (output) Lower bounds for the variables (arcs) in the embedded network.

bux (output) Upper bounds for the variables (arcs) in the embedded network.

from (output) Defines the origins of the arcs in the embedded network.

to (output) Defines the destinations of the arcs in the embedded network.

numcon (output) Number of network constraints (nodes) in the embedded network.

numvar (output) Number of network variables (arcs) in the embedded network.

Description: Uses a heuristic to find an embedded network in the problem specified in `task`.

The returned network is a pure network flow problem and can be solved with a direct call to the network optimizer `Task.netoptimize`.

Each arc a in the network corresponds to a scaled subset of elements in column $j = \text{netvar}[a]$ from the constraint matrix A stored in `task`. Each node n in the network corresponds to a scaled subset of elements in constraint $i = \text{netcon}[n]$ from A . Data structures for network problems is explained in 5.2 and 6.2. The relation between A and the extracted embedded network can be explained as follows:

$$- A_{\text{netcon}[\text{from}[a]], \text{netvar}[a]} * \text{scalcon}[\text{netcon}[\text{from}[a]]] * \text{scalvar}[\text{netvar}[a]] = -1$$

- $A_{\text{netcon}[\text{to}[a]], \text{netvar}[a]} * \text{scalcon}[\text{netcon}[\text{to}[a]]] * \text{scalvar}[\text{netvar}[a]] = 1$
- The scaled matrix has at most two non-zeroes in each column in **netvar** over the indexes in **netcon** (i.e defines a pure network flow matrix).

Please note if a column $j = \text{netvar}[a]$ is only represented by one non-zero in the embedded network, then either $\text{from}[a] = \text{netcon}$ or $\text{to}[a] = \text{netcon}$.

See also:

Task.netoptimize Optimizes a pure network flow problem.

- `mosek.Task.netoptimize`

Syntax:

```
prosta,solsta = netoptimize (
    array(double) cc,
    array(double) cx,
    array(boundkey) bkc,
    array(double) blc,
    array(double) buc,
    array(boundkey) bkc,
    array(double) blx,
    array(double) bux,
    array(int) from,
    array(int) to,
    int hotstart,
    array(stakey) skc,
    array(stakey) skx,
    array(double) xc,
    array(double) xx,
    array(double) y,
    array(double) slc,
    array(double) suc,
    array(double) slx,
    array(double) sux)
```

cc (input) Linear terms of the objective for constraints (nodes) as a dense vector.

cx (input) Linear terms of the objective for variables (arcs) as a dense vector.

bkc (input) Bound keys for the constraints (nodes).

blc (input) Lower bounds for the constraints (nodes).

buc (input) Upper bounds for the constraints (nodes).

bkc (input) Bound keys for the variables (arcs).

blx (input) Lower bounds for the variables (arcs).

bux (input) Upper bounds for the variables (arcs).

from (input) Defines the origins of the arcs in the network.

to (input) Defines the destinations of the arcs in the network.

hotstart (input) If zero the network optimizer will not use hot-starts, if non-zero a solution must be defined in the solution variables below, which will be used to hot-start the network optimizer.

skc (input/output) Status keys for the constraints (nodes).

skx (input/output) Status keys for the variables (arcs).

xc (input/output) Primal constraint solution (nodes).

xx (input/output) Primal variable solution (arcs).

y (input/output) Vector of dual variables corresponding to the constraints (nodes).

slc (input/output) Dual variables corresponding to the lower bounds on the constraints (nodes).

suc (input/output) Dual variables corresponding to the upper bounds on the constraints (nodes).

slx (input/output) Dual variables corresponding to the lower bounds on the constraints (arcs).

sux (input/output) Dual variables corresponding to the upper bounds on the constraints (arcs).

prosta (output) Problem status.

solsta (output) Solution status.

Description: Uses the network optimizer to solve the given network problem. The problem must be a pure network flow problem. If **hotstart** is zero the network optimizer will not use hot-starts, if non-zero a solution must be defined in the solution variables **skc, skx, xc, xx, y, slc, suc, slx** and **sux**, which will be used to hot-start the network optimizer. Please note **task** only acts as a dummy task, where parameters and streams can be set for the network optimizer. No other data in **task** is used.

See also:

Task.netextraction Finds embedded network structure.

- `mosek.Task.optimizeconcurrent`

Syntax:

`optimizeconcurrent (array(Task) taskarray)`

taskarray (input) An array of **num** tasks.

Description: Solves several instances of the same problem in parallel, with unique parameter settings for each task. The argument **task** contains the problem to be solved. **taskarray** is a pointer to an array of **num** empty tasks. The task **task** and the **num** tasks pointed to by **taskarray** are solved in parallel. That is **num** + 1 threads are started with one optimizer in each. Each of the tasks can be initialized with different parameters, e.g different selection of solver.

All the concurrently running tasks are stopped when the optimizer successfully terminates for one of the tasks. After the function returns **task** contains the solution found by the task that finished first.

After **Task.optimizeconcurrent** returns **task** holds the optimal solution of the task which finished first. If all the concurrent optimizations finished without providing an optimal solution the error code from the solution of the task **task** is returned.

In summary a call to `Task.optimizeconcurrent` does the following:

1. All data except task parameters (`mosek.iparam`, `mosek.dparam` and `mosek.sparam`) in `task` is copied to each of the tasks in `taskarray`. In particular this means that any solution in `task` is copied to the other tasks. Call-back functions are not copied.
2. The tasks `task` and the `num` tasks in `taskarray` are started in parallel.
3. When a task finishes providing an optimal solution (or a certificate of infeasibility) its solution is copied to `task` and all other tasks are stopped.

For an explained code example see Section 8.6.4.

- `mosek.Task.optimizersummary`

Syntax:

```
optimizersummary (mosek.streamtype whichstream)
whichstream (input) Index of the stream.
```

Description: Prints a short summary with optimizer statistics for last optimization.

- `mosek.Task.optimize`

Syntax:

```
termcode = optimize ()
termcode (output) Is either mosek.rescode.ok or a termination response code.
```

Description: 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 `mosek.iparam.optimizer`.

See also:

`Task.optimizeconcurrent` Optimize a given task with several optimizers concurrently.
`Task.getsolution` Obtains the complete solution.
`Task.getsolutioni` Obtains the solution for a single constraint or variable.
`Task.getsolutioninf` Obtains information about a solution.
`mosek.iparam.optimizer`

- `mosek.Task.primalsensitivity`

Syntax:

```
primalsensitivity (
    array(int) subi,
    array(mark) marki,
    array(int) subj,
    array(mark) markj,
    array(double) leftpricei,
    array(double) rightpricei,
    array(double) leftrangei,
```

```

    array(double) rightrangei,
    array(double) leftpricej,
    array(double) rightpricej,
    array(double) leftrangej,
    array(double) rightrangej)

```

subi (input) Indexes of bounds on constraints to analyze.

marki (input) The value of `marki[i]` specifies for which bound (upper or lower) on constraint `subi[i]` sensitivity analysis should be performed.

subj (input) Indexes of bounds on variables to analyze.

markj (input) The value of `markj[j]` specifies for which bound (upper or lower) on variable `subj[j]` sensitivity analysis should be performed.

leftpricei (output) `leftpricei[i]` is the left shadow price for the upper/lower bound (indicated by `marki[i]`) of the constraint with index `subi[i]`.

rightpricei (output) `rightpricei[i]` is the right shadow price for the upper/lower bound (indicated by `marki[i]`) of the constraint with index `subi[i]`.

leftrangei (output) `leftrangei[i]` is the left range for the upper/lower bound (indicated by `marki[i]`) of the constraint with index `subi[i]`.

rightrangei (output) `rightrangei[i]` is the right range for the upper/lower bound (indicated by `marki[i]`) of the constraint with index `subi[i]`.

leftpricej (output) `leftpricej[j]` is the left shadow price for the upper/lower bound (indicated by `marki[j]`) on variable `subj[j]`.

rightpricej (output) `rightpricej[j]` is the right shadow price for the upper/lower bound (indicated by `marki[j]`) on variable `subj[j]`.

leftrangej (output) `leftrangej[j]` is the left range for the upper/lower bound (indicated by `marki[j]`) on variable `subj[j]`.

rightrangej (output) `rightrangej[j]` is the right range for the upper/lower bound (indicated by `marki[j]`) on variable `subj[j]`.

Description: Calculates sensitivity information for bounds on variables and constraints.

For details on sensitivity analysis and the definitions of *shadow price* and *linearity interval* see chapter 12.

The constraints for which sensitivity analysis is performed are given by the data structures:

1. `subi` Index of constraint to analyze.
2. `marki` Indicate for which bound of constraint `subi[i]` sensitivity analysis is performed. If `marki[i] = mosek.mark.up` the upper bound of constraint `subi[i]` is analyzed, and if `marki[i] = mosek.mark.lo` the lower bound is analyzed. If `subi[i]` is an equality constraint, either `mosek.mark.lo` or `mosek.mark.up` can be used to select the constraint for sensitivity analysis.

Consider the problem:

$$\begin{aligned}
 &\text{minimize} && x_1 + x_2 \\
 &\text{subject to} && -1 \leq x_1 - x_2 \leq 1, \\
 &&& x_1 = 0, \\
 &&& x_1 \geq 0, x_2 \geq 0
 \end{aligned} \tag{14.24}$$

Suppose that

```

numi = 1;
subi = [0];
marki = [mosek.mark.up]

```

then

`leftpricei[0]`, `rightpricei[0]`, `leftrangei[0]` and `rightrangei[0]` will contain the sensitivity information for the upper bound on constraint 0 given by the expression:

$$x_1 - x_2 \leq 1 \quad (14.25)$$

Similarly, the variables for which to perform sensitivity analysis are given by the structures:

1. `subj` Index of variables to analyze.
2. `markj` Indicate for which bound of variable `subi[j]` sensitivity analysis is performed. If `markj[j] = mosek.mark.up` the upper bound of constraint `subi[j]` is analyzed, and if `markj[j] = mosek.mark.lo` the lower bound is analyzed. If `subi[j]` is an equality constraint, either `mosek.mark.lo` or `mosek.mark.up` can be used to select the constraint for sensitivity analysis.

For an example, please see Section 12.5.

The type of sensitivity analysis to be performed (basis or optimal partition) is controlled by the parameter `mosek.iparam.sensitivity_type`.

See also:

`Task.dualsensitivity` Performs sensitivity analysis on objective coefficients.

`Task.sensitivityreport` Creates a sensitivity report.

`mosek.iparam.sensitivity_type`

`mosek.iparam.log_sensitivity`

`mosek.iparam.log_sensitivity_opt`

- `mosek.Task.printdata`

Syntax:

```

printdata (
    mosek.streamtype whichstream,
    int firsti,
    int lasti,
    int firstj,
    int lastj,
    int firstk,
    int lastk,
    int c,
    int qo,
    int a,
    int qc,
    int bc,
    int bx,
    int vartype,
    int cones)

```

whichstream (input) Index of the stream.
firsti (input) Index of first constraint for which data should be printed.
lasti (input) Index of last constraint plus 1 for which data should be printed.
firstj (input) Index of first variable for which data should be printed.
lastj (input) Index of last variable plus 1 for which data should be printed.
firstk (input) Index of first cone for which data should be printed.
lastk (input) Index of last cone plus 1 for which data should be printed.
c (input) If non-zero c is printed.
qo (input) If non-zero Q^o is printed.
a (input) If non-zero A is printed.
qc (input) If non-zero Q^k is printed for the relevant constraints.
bc (input) If non-zero the constraints bounds are printed.
bx (input) If non-zero the variable bounds are printed.
vartype (input) If non-zero the variable types are printed.
cones (input) If non-zero the conic data is printed.

Description: Prints a part of the problem data to a stream. This function is normally used for debugging purposes only, e.g. to verify that the correct data has been inputted.

- `mosek.Task.printparam`

Syntax:

```
printparam ()
```

Description: Prints the current parameter settings to the message stream.

- `mosek.Task.putaij`

Syntax:

```
putaij (
    int i,
    int j,
    double aij)
```

i (input) Index of the constraint in which the change should occur.

j (input) Index of the variable in which the change should occur.

aij (input) New coefficient for $a_{i,j}$.

Description: Changes a coefficient in A using the method

$$a_{ij} = \text{aij}.$$

See also:

Task.putavec Replaces all elements in one row or column of the linear coefficient matrix.

Task.putaij Changes a single value in the linear coefficient matrix.

- `mosek.Task.putaijlist`

Syntax:

```
putaijlist (
    array(int) subi,
    array(int) subj,
    array(double) valij)
```

subi (input) Constraint indexes in which the change should occur.

subj (input) Variable indexes in which the change should occur.

valij (input) New coefficient values for $a_{i,j}$.

Description: Changes one or more coefficients in A using the method

$$a_{\text{subi}[k], \text{subj}[k]} = \text{valij}[k], \quad k = 0, \dots, \text{num} - 1.$$

If the same $a_{i,j}$ entry appears multiple times only the last one will be used.

See also:

Task.putavec Replaces all elements in one row or column of the linear coefficient matrix.

Task.putaij Changes a single value in the linear coefficient matrix.

- `mosek.Task.putavec`

Syntax:

```
putavec (
    mosek.accmode accmode,
    int i,
    array(int) asub,
    array(double) aval)
```

accmode (input) Defines whether to replace a column or a row.

i (input) If `accmode` equals `mosek.accmode.con`, then i is a constraint index. Otherwise it is a column index.

asub (input) Index of the $a_{i,j}$ values that should be changed.

aval (input) New $a_{i,j}$ values.

Description: Replaces all elements in one row or column of A .

Assuming that there are no duplicate subscripts in `asub`, assignment is performed as follows:

- If `accmode` is `mosek.accmode.con`, then

$$a_{i, \text{asub}[k]} = \text{aval}[k], \quad k = 0, \dots, \text{nzi} - 1$$

and all other $a_{i,\cdot} = 0$.

- If `accmode` is `mosek.accmode.var`, then

$$a_{\text{asub}[k], i} = \text{aval}[k], \quad k = 0, \dots, \text{nzi} - 1$$

and all other $a_{\cdot, i} = 0$.

If `asub` contains duplicates, the corresponding coefficients will be added together.

For an explanation of the meaning of `ptrb` and `ptre` see Section 5.8.3.2.

See also:

Task.putavec Replaces all elements in one row or column of the linear coefficient matrix.

Task.putaij Changes a single value in the linear coefficient matrix.

- `mosek.Task.putaveclist`

Syntax:

```
putaveclist (
    mosek.accmode accmode,
    array(int) sub,
    array(long) ptrb,
    array(long) ptre,
    array(int) asub,
    array(double) aval)
```

accmode (input) Defines if operations are performed row-wise (constraint-oriented) or column-wise (variable-oriented).

sub (input) Indexes of rows or columns that should be replaced. **sub** should not contain duplicate values.

ptrb (input) Array of pointers to the first element in the rows or columns stored in **asub** and **aval**. For an explanation of the meaning of **ptrb** see Section 5.8.3.2.

ptre (input) Array of pointers to the last element plus one in the rows or columns stored in **asub** and **aval**. For an explanation of the meaning of **ptre** see Section 5.8.3.2.

asub (input) If **accmode** is `mosek.accmode.con`, then **asub** contains the new variable indexes, otherwise it contains the new constraint indexes.

aval (input) Coefficient values. See (5.33) and Section 5.8.3.

Description: Replaces all elements in a set of rows or columns of A .

The elements are replaced as follows.

- If **accmode** is `mosek.accmode.con`, then for $i = 0, \dots, \text{num} - 1$

$$a_{\text{sub}[i], \text{asub}[k]} = \text{aval}[k], \quad k = \text{aptrb}[i], \dots, \text{aptre}[i] - 1.$$

- If **accmode** is `mosek.accmode.var`, then for $i = 0, \dots, \text{num} - 1$

$$a_{\text{asub}[k], \text{sub}[i]} = \text{aval}[k], \quad k = \text{aptrb}[i], \dots, \text{aptre}[i] - 1.$$

If the same row or column appears multiple times only the last one will be used.

See also:

Task.putavec Replaces all elements in one row or column of the linear coefficient matrix.

- `mosek.Task.putbound`

Syntax:

```

putbound (
    mosek.accmode accmode,
    int i,
    mosek.boundkey bk,
    double bl,
    double bu)

```

accmode (input) Defines whether the bound for a constraint or a variable is changed.

i (input) Index of the constraint or variable.

bk (input) New bound key.

bl (input) New lower bound.

bu (input) New upper bound.

Description: Changes the bounds for either one constraint or one variable.

If the a bound value specified is numerically larger than `mosek.dparam.data_tol_bound_inf` it is considered infinite and the bound key is changed accordingly. If a bound value is numerically larger than `mosek.dparam.data_tol_bound_wrn`, a warning will be displayed, but the bound is inputted as specified.

See also:

`Task.putboundlist` Changes the bounds of constraints or variables.

- `mosek.Task.putboundlist`

Syntax:

```

putboundlist (
    mosek.accmode accmode,
    array(int) sub,
    array(boundkey) bk,
    array(double) bl,
    array(double) bu)

```

accmode (input) Defines whether bounds for constraints (`mosek.accmode.con`) or variables (`mosek.accmode.var`) are changed.

sub (input) Subscripts of the bounds that should be changed.

bk (input) Constraint or variable index `sub[t]` is assigned the bound key `bk[t]`.

bl (input) Constraint or variable index `sub[t]` is assigned the lower bound `bl[t]`.

bu (input) Constraint or variable index `sub[t]` is assigned the upper bound `bu[t]`.

Description: Changes the bounds for either some constraints or variables. If multiple bound changes are specified for a constraint or a variable, only the last change takes effect.

See also:

`Task.putbound` Changes the bound for either one constraint or one variable.

`mosek.dparam.data_tol_bound_inf`

`mosek.dparam.data_tol_bound_wrn`

- `mosek.Task.putboundslice`

Syntax:

```
putboundslice (
    mosek.accmode con,
    int first,
    int last,
    array(boundkey) bk,
    array(double) bl,
    array(double) bu)
```

con (input) Defines whether bounds for constraints (**mosek.accmode.con**) or variables (**mosek.accmode.var**) are changed.

first (input) First index in the sequence.

last (input) Last index plus 1 in the sequence.

bk (input) Bound keys.

bl (input) Values for lower bounds.

bu (input) Values for upper bounds.

Description: Changes the bounds for a sequence of variables or constraints.

See also:

Task.putbound Changes the bound for either one constraint or one variable.

mosek.dparam.data_tol_bound_inf

mosek.dparam.data_tol_bound_wrn

- **mosek.Task.putcfix**

Syntax:

```
putcfix (double cfix)
cfix (input) Fixed term in the objective.
```

Description: Replaces the fixed term in the objective by a new one.

- **mosek.Task.putcj**

Syntax:

```
putcj (
    int j,
    double cj)
j (input) Index of the variable for which  $c$  should be changed.
cj (input) New value of  $c_j$ .
```

Description: Modifies one coefficient in the linear objective vector c , i.e.

$$c_j = cj.$$

- **mosek.Task.putclist**

Syntax:

```

putclist (
    array(int) subj,
    array(double) val)

```

subj (input) Index of variables for which c should be changed.

val (input) New numerical values for coefficients in c that should be modified.

Description: Modifies elements 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.

- `mosek.Task.putcone`

Syntax:

```

putcone (
    int k,
    mosek.conetype conetype,
    double coneapar,
    array(int) submem)

```

k (input) Index of the cone.

conetype (input) Specifies the type of the cone.

coneapar (input) This argument is currently not used. Can be set to 0.0.

submem (input) Variable subscripts of the members in the cone.

Description: Replaces a conic constraint.

- `mosek.Task.putdoupam`

Syntax:

```

putdoupam (
    mosek.dparam param,
    double parvalue)

```

param (input) Which parameter.

parvalue (input) Parameter value.

Description: Sets the value of a double parameter.

- `mosek.Task.putintparam`

Syntax:

```

putintparam (
    mosek.iparam param,
    int parvalue)

```

param (input) Which parameter.

parvalue (input) Parameter value.

Description: Sets the value of an integer parameter.

- `mosek.Task.putmaxnumanz`

Syntax:

```
putmaxnumanz (long maxnumanz)
maxnumanz (input) New size of the storage reserved for storing A.
```

Description: The function changes the size of the preallocated storage for linear coefficients.

See also:

`mosek.iinfitem.sto_num_a_realloc`

- `mosek.Task.putmaxnumcon`

Syntax:

```
putmaxnumcon (int maxnumcon)
maxnumcon (input) Number of preallocated constraints in the optimization task.
```

Description: 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.

- `mosek.Task.putmaxnumcone`

Syntax:

```
putmaxnumcone (int maxnumcone)
maxnumcone (input) Number of preallocated conic constraints in the optimization task.
```

Description: 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 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.

- `mosek.Task.putmaxnumqnz`

Syntax:

```
putmaxnumqnz (long maxnumqnz)
maxnumqnz (input) Number of non-zero elements preallocated in quadratic coefficient matrices.
```

Description: 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 never mandatory to call this function, since its only function is to give a hint of the amount of data to preallocate for efficiency reasons.

- `mosek.Task.putmaxnumvar`

Syntax:

```
putmaxnumvar (int maxnumvar)
```

`maxnumvar` (**input**) Number of preallocated variables in the optimization task.

Description: 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 never mandatory to call this function, since its only function is to give a hint of 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.

- `mosek.Task.putnadouparam`

Syntax:

```
putnadouparam (
    str paramname,
    double parvalue)
```

`paramname` (**input**) Name of a parameter.

`parvalue` (**input**) Parameter value.

Description: Sets the value of a named double parameter.

- `mosek.Task.putnaintparam`

Syntax:

```
putnaintparam (
    str paramname,
    int parvalue)
```

`paramname` (**input**) Name of a parameter.

`parvalue` (**input**) Parameter value.

Description: Sets the value of a named integer parameter.

- `mosek.Task.putname`

Syntax:

```

putname (
    mosek.problemitem whichitem,
    int i,
    str name)

```

whichitem (input) Problem item, i.e. a cone, a variable or a constraint name..

i (input) Index.

name (input) New name to be assigned to the item.

Description: Assigns the name defined by **name** to a problem item (a variable, a constraint or a cone).

- `mosek.Task.putnastrparam`

Syntax:

```

putnastrparam (
    str paramname,
    str parvalue)

```

paramname (input) Name of a parameter.

parvalue (input) Parameter value.

Description: Sets the value of a named string parameter.

- `mosek.Task.putobjname`

Syntax:

```

putobjname (str objname)
objname (input) Name of the objective.

```

Description: Assigns the name given by **objname** to the objective function.

- `mosek.Task.putobjsense`

Syntax:

```

putobjsense (mosek.objsense sense)
sense (input) The objective sense of the task. The values mosek.objsense.maximize and
mosek.objsense.minimize means that the the problem is maximized or minimized
respectively. The value mosek.objsense.undefined means that the objective sense is
taken from the parameter mosek.iparam.objective_sense.

```

Description: Sets the objective sense of the task.

See also:

`Task.getobjsense` Gets the objective sense.

- `mosek.Task.putparam`

Syntax:

```

putparam (
    str parname,
    str parvalue)
parname (input) Parameter name.
parvalue (input) Parameter value.

```

Description: Checks if a **parname** is valid parameter name. If it is, the parameter is assigned the value specified by **parvalue**.

- `mosek.Task.putqcon`

Syntax:

```

putqcon (
    array(int) qcsbk,
    array(int) qcsubi,
    array(int) qcsubj,
    array(double) qcval)
qcsbk (input)  $k$  subscripts for  $q_{ij}^k$ . See (5.32).
qcsubi (input)  $i$  subscripts for  $q_{ij}^k$ . See (5.32).
qcsubj (input)  $j$  subscripts for  $q_{ij}^k$ . See (5.32).
qcval (input) Numerical value for  $q_{ij}^k$ .

```

Description: Replaces all quadratic entries in the constraints. Consider constraints on the form:

$$l_k^c \leq \frac{1}{2} \sum_{i=0}^{\text{numvar}-1} \sum_{j=0}^{\text{numvar}-1} q_{ij}^k x_i x_j + \sum_{j=0}^{\text{numvar}-1} a_{kj} x_j \leq u_k^c, \quad k = 0, \dots, m-1. \quad (14.26)$$

The function assigns values to q such that:

$$q_{\text{qcsubi}[t], \text{qcsubj}[t]}^{\text{qcsbk}[t]} = \text{qcval}[t], \quad t = 0, \dots, \text{numqcnz} - 1. \quad (14.27)$$

and

$$q_{\text{qcsubj}[t], \text{qcsubi}[t]}^{\text{qcsbk}[t]} = \text{qcval}[t], \quad t = 0, \dots, \text{numqcnz} - 1. \quad (14.28)$$

Values not assigned are set to zero.

Please note that duplicate entries are added together.

See also:

Task.putqconk Replaces all quadratic terms in a single constraint.

- `mosek.Task.putqconk`

Syntax:

```

putqconk (
    int k,
    array(int) qcsubi,
    array(int) qcsubj,
    array(double) qcval)

```

k (input) The constraint in which the new Q elements are inserted.

qcsubi (input) i subscripts for q_{ij}^k . See (5.32).

qcsubj (input) j subscripts for q_{ij}^k . See (5.32).

qcval (input) Numerical value for q_{ij}^k .

Description: Replaces all the quadratic entries in one constraint k of the form:

$$l_k^c \leq \frac{1}{2} \sum_{i=0}^{\text{numvar}-1} \sum_{j=0}^{\text{numvar}-1} q_{ij}^k x_i x_j + \sum_{j=0}^{\text{numvar}-1} a_{kj} x_j \leq u_k^c. \quad (14.29)$$

It is assumed that Q^k is symmetric, i.e. $q_{ij}^k = q_{ji}^k$, and therefore, only the values of q_{ij}^k for which $i \geq j$ should be inputted to MOSEK. To be precise, MOSEK uses the following procedure

1. $Q^k = 0$
2. for $t = 0$ to $\text{numqonz} - 1$
3. $q_{\text{qcsubi}[t], \text{qcsubj}[t]}^k = q_{\text{qcsubi}[t], \text{qcsubj}[t]}^k + \text{qcval}[t]$
3. $q_{\text{qcsubj}[t], \text{qcsubi}[t]}^k = q_{\text{qcsubj}[t], \text{qcsubi}[t]}^k + \text{qcval}[t]$

Please note that:

- For large problems it is essential for the efficiency that the function `Task.putmaxnumqnz64` is employed to specify an appropriate `maxnumqnz`.
- Only the lower triangular part should be specified because Q^k is symmetric. Specifying values for q_{ij}^k 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. Hence, it is recommended not to specify the same element multiple times in `qosubi`, `qosubj`, and `qoval`.

For a code example see Section 5.3.2.

See also:

`Task.putqcon` Replaces all quadratic terms in constraints.

- `mosek.Task.putqobj`

Syntax:

```
putqobj (
    array(int) qosubi,
    array(int) qosubj,
    array(double) qoval)
```

qosubi (input) i subscript for q_{ij}^o .

qosubj (input) j subscript for q_{ij}^o .

qoval (input) Numerical value for q_{ij}^o .

Description: Replaces all the quadratic terms in the objective

$$\frac{1}{2} \sum_{i=0}^{\text{numvar}-1} \sum_{j=0}^{\text{numvar}-1} q_{ij}^o x_i x_j + \sum_{j=0}^{\text{numvar}-1} c_j x_j + c^f. \quad (14.30)$$

It is assumed that Q^o is symmetric, i.e. $q_{ij}^o = q_{ji}^o$, and therefore, only the values of q_{ij}^o for which $i \geq j$ should be specified. To be precise, MOSEK uses the following procedure

1. $Q^o = 0$
2. for $t = 0$ to $\text{numqonz} - 1$
3. $q_{\text{qosubi}[t], \text{qosubj}[t]}^o = q_{\text{qosubi}[t], \text{qosubj}[t]}^o + \text{qoval}[t]$
3. $q_{\text{qosubj}[t], \text{qosubi}[t]}^o = q_{\text{qosubj}[t], \text{qosubi}[t]}^o + \text{qoval}[t]$

Please note that:

- Only the lower triangular part should be specified because Q^o is symmetric. Specifying values for q_{ij}^o 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 entries are added to together.

For a code example see Section 5.3.1.

- `mosek.Task.putqobjij`

Syntax:

```
putqobjij (
    int i,
    int j,
    double qoij)
```

i (input) Row index for the coefficient to be replaced.

j (input) Column index for the coefficient to be replaced.

qoij (input) The new value for q_{ij}^o .

Description: Replaces one coefficient in the quadratic term in the objective. The function performs the assignment

$$q_{ij}^o = \text{qoij}.$$

Only the elements in the lower triangular part are accepted. Setting q_{ij} with $j > i$ will cause an error.

Please note that replacing all quadratic element, one at a time, is more computationally expensive than replacing all elements at once. Use `Task.putqobj` instead whenever possible.

- `mosek.Task.putsolution`

Syntax:

```
putsolution (
    mosek.soltype whichsol,
    array(stakey) skc,
    array(stakey) skx,
    array(stakey) skn,
    array(double) xc,
    array(double) xx,
    array(double) y,
    array(double) slc,
    array(double) suc,
    array(double) slx,
    array(double) sux,
    array(double) snx)
```

whichsol (input) Selects a solution.

skc (input) Status keys for the constraints.

skx (input) Status keys for the variables.

skn (input) Status keys for the conic constraints.

xc (input) Primal constraint solution.

xx (input) Primal variable solution (x).

y (input) Vector of dual variables corresponding to the constraints.

slc (input) Dual variables corresponding to the lower bounds on the constraints (s_l^c).

suc (input) Dual variables corresponding to the upper bounds on the constraints (s_u^c).

slx (input) Dual variables corresponding to the lower bounds on the variables (s_l^x).

sux (input) Dual variables corresponding to the upper bounds on the variables (appears as s_u^x).

snx (input) Dual variables corresponding to the conic constraints on the variables (s_n^x).

Description: Inserts a solution into the task.

- `mosek.Task.putsolutioni`

Syntax:

```
putsolutioni (
    mosek.accmode accmode,
    int i,
    mosek.soltype whichsol,
    mosek.stakey sk,
    double x,
    double sl,
    double su,
    double sn)
```

accmode (input) If set to `mosek.accmode.con` the solution information for a constraint is modified. Otherwise for a variable.

i (input) Index of the constraint or variable.

whichsol (**input**) Selects a solution.
sk (**input**) Status key of the constraint or variable.
x (**input**) Solution value of the primal constraint or variable.
sl (**input**) Solution value of the dual variable associated with the lower bound.
su (**input**) Solution value of the dual variable associated with the upper bound.
sn (**input**) Solution value of the dual variable associated with the cone constraint.

Description: Sets the primal and dual solution information for a single constraint or variable.

To define a solution or a significant part of a solution, first call the **Task.makesolutionstatusunknown** function, then for each relevant constraint and variable call **Task.putsolutioni** to set the solution information.

See also:

Task.makesolutionstatusunknown Sets the solution status to unknown.

- **mosek.Task.putsolutionyi**

Syntax:

```
putsolutionyi (
    int i,
    mosek.soltype whichsol,
    double y)

i (input) Index of the dual variable.
whichsol (input) Selects a solution.
y (input) Solution value of the dual variable.
```

Description: Inputs the dual variable of a solution.

See also:

Task.makesolutionstatusunknown Sets the solution status to unknown.

Task.putsolutioni Sets the primal and dual solution information for a single constraint or variable.

- **mosek.Task.putstrparam**

Syntax:

```
putstrparam (
    mosek.sparam param,
    str parvalue)

param (input) Which parameter.
parvalue (input) Parameter value.
```

Description: Sets the value of a string parameter.

- **mosek.Task.puttaskname**

Syntax:

```
puttaskname (str taskname)
```

taskname (input) Name assigned to the task.

Description: Assigns the name **taskname** to the task.

- `mosek.Task.putvarbranchorder`

Syntax:

```
putvarbranchorder (
    int j,
    int priority,
    mosek.branchdir direction)
```

j (input) Index of the variable.

priority (input) The branching priority that should be assigned to variable *j*.

direction (input) Specifies the preferred branching direction for variable *j*.

Description: The purpose of the function is to assign a branching priority and direction. The higher priority that is assigned to an integer variable the earlier the mixed integer optimizer will branch on the variable. The branching direction controls if the optimizer branches up or down on the variable.

- `mosek.Task.putvartype`

Syntax:

```
putvartype (
    int j,
    mosek.variabletype vartype)
```

j (input) Index of the variable.

vartype (input) The new variable type.

Description: Sets the variable type of one variable.

See also:

Task.putvartypelist Sets the variable type for one or more variables.

- `mosek.Task.putvartypelist`

Syntax:

```
putvartypelist (
    array(int) subj,
    array(variabletype) vartype)
```

subj (input) A list of variable indexes for which the variable type should be changed.

vartype (input) A list of variable types that should be assigned to the variables specified by **subj**. See section 17.55 for the possible values of **vartype**.

Description: Sets the variable type for one or more variables, i.e. variable number **subj**[*k*] is assigned the variable type **vartype**[*k*].

If the same index is specified multiple times in **subj** only the last entry takes effect.

See also:

`Task.putvartype` Sets the variable type of one variable.

- `mosek.Task.readbranchpriorities`

Syntax:

`readbranchpriorities (str filename)`

`filename (input)` Data is read from the file `filename`.

Description: Reads branching priority data from a file.

See also:

`Task.writebranchpriorities` Writes branching priority data to a file.

- `mosek.Task.readdata`

Syntax:

`readdata (str filename)`

`filename (input)` Data is read from the file `filename` if it is a nonempty string. Otherwise data is read from the file specified by `mosek.sparam.data_file_name`.

Description: Reads an optimization data and associated data from a file.

The data file format is determined by the `mosek.iparam.read_data_format` parameter. By default the parameter has the value `mosek.dataformat.extension` indicating that the extension of the input file should determine the file type, where the extension is interpreted as follows:

- “.lp” and “.lp.gz” are interpreted as an LP file and a compressed LP file respectively.
- “.opf” and “.opf.gz” are interpreted as an OPF file and a compressed OPF file respectively.
- “.mps” and “.mps.gz” are interpreted as an MPS file and a compressed MPS file respectively.
- “.mbt” and “.mbt.gz” are interpreted as an MBT file and a compressed MBT file respectively.

See also:

`Task.writedata` Writes problem data to a file.

`mosek.iparam.read_data_format`

- `mosek.Task.readparamfile`

Syntax:

`readparamfile ()`

Description: Reads a parameter file.

- `mosek.Task.readsolution`

Syntax:

```

readsolution (
    mosek.soltype whichsol,
    str filename)

whichsol (input) Selects a solution.
filename (input) A valid file name.

```

Description: Reads a solution file and inserts the solution into the solution **whichsol**.

- `mosek.Task.readsummary`

Syntax:

```

readsummary (mosek.streamtype whichstream)

whichstream (input) Index of the stream.

```

Description: Prints a short summary of last file that was read.

- `mosek.Task.relaxprimal`

Syntax:

```

relaxedtask = relaxprimal (
    array(double) wlc,
    array(double) wuc,
    array(double) wlx,
    array(double) wux)

```

wlc (**input/output**) Weights associated with lower bounds on the activity of constraints. If negative, the bound is strictly enforced, i.e. if $(w_l^c)_i < 0$, then $(v_l^c)_i$ is fixed to zero. On return **wlc**[i] contains the relaxed bound.

wuc (**input/output**) Weights associated with upper bounds on the activity of constraints. If negative, the bound is strictly enforced, i.e. if $(w_u^c)_i < 0$, then $(v_u^c)_i$ is fixed to zero. On return **wuc**[i] contains the relaxed bound.

wlx (**input/output**) Weights associated with lower bounds on the activity of variables. If negative, the bound is strictly enforced, i.e. if $(w_l^x)_j < 0$ then $(v_l^x)_j$ is fixed to zero. On return **wlx**[i] contains the relaxed bound.

wux (**input/output**) Weights associated with upper bounds on the activity of variables. If negative, the bound is strictly enforced, i.e. if $(w_u^x)_j < 0$ then $(v_u^x)_j$ is fixed to zero. On return **wux**[i] contains the relaxed bound.

relaxedtask (**output**) The returned task.

Description: Creates a problem that computes the minimal (weighted) relaxation of the bounds that will make an infeasible problem feasible.

Given an existing task describing the problem

$$\begin{aligned}
 &\text{minimize} && c^T x \\
 &\text{subject to} && l^c \leq Ax \leq u^c, \\
 &&& l^x \leq x \leq u^x,
 \end{aligned} \tag{14.31}$$

the function forms a new task **relaxedtask** having the form

$$\begin{aligned}
 & \text{minimize} && p \\
 & \text{subject to} && \begin{aligned}
 l^c & \leq Ax + v_l^c - v_u^c & \leq u^c, \\
 l^x & \leq x + v_l^x - v_u^x & \leq u^x, \\
 (w_l^c)^T v_l^c + (w_u^c)^T v_u^c + (w_l^x)^T v_l^x + (w_u^x)^T v_u^x - p & \leq 0, \\
 v_l^c, v_u^c, v_l^x, v_u^x & \geq 0.
 \end{aligned}
 \end{aligned} \tag{14.32}$$

Hence, the function adds so-called elasticity variables to all the constraints which relax the constraints, for instance $(v_l^c)_i$ and $(v_u^c)_i$ relax $(l^c)_i$ and $(u^c)_i$ respectively. It should be obvious that (14.32) is feasible. Moreover, the function adds the constraint

$$(w_l^c)^T v_l^c + (w_u^c)^T v_u^c + (w_l^x)^T v_l^x + (w_u^x)^T v_u^x - p \leq 0$$

to the problem which makes the variable p bigger than the total weighted sum of the relaxation to the bounds. w_l^c , w_u^c , w_l^x and w_u^x are user-defined weights which normally should be nonnegative. If a weight is negative, then the corresponding elasticity variable is fixed to zero.

Hence, when the problem (14.32) is optimized, the weighted minimal change to the bounds such that the problem is feasible is computed.

One can specify that a bound should be strictly enforced by assigning a negative value to the corresponding weight, i.e if $(w_l^c)_i < 0$ then $(v_l^c)_i$ is fixed to zero.

Now let p^* be the optimal objective value to (14.32), then a natural thing to do is to solve the optimization problem

$$\begin{aligned}
 & \text{minimize} && c^T x \\
 & \text{subject to} && \begin{aligned}
 l^c & \leq Ax + v_l^c - v_u^c & \leq u^c, \\
 l^x & \leq x + v_l^x - v_u^x & \leq u^x, \\
 (w_l^c)^T v_l^c + (w_u^c)^T v_u^c + (w_l^x)^T v_l^x + (w_u^x)^T v_u^x - p & \leq 0, \\
 p & = p^*, \\
 v_l^c, v_u^c, v_l^x, v_u^x & \geq 0,
 \end{aligned}
 \end{aligned} \tag{14.33}$$

where the original objective function is minimized subject to the constraint that the total weighted relaxation is minimal.

The parameter **mosek.iparam.feasrepair_optimize** controls whether the function returns the problem (14.32) or the problem (14.33). The parameter can take one of the following values.

mosek.feasrepairtype.optimize_none : The returned task **relaxedtask** contains problem (14.32) and is not optimized.

mosek.feasrepairtype.optimize_penalty : The returned task **relaxedtask** contains problem (14.32) and is optimized.

mosek.feasrepairtype.optimize_combined : The returned task **relaxedtask** contains problem (14.33) and is optimized.

Please note that the v variables are appended to the x variables ordered as

$$(v_u^c)_1, (v_l^c)_1, (v_u^c)_2, (v_l^c)_2, \dots, (v_u^c)_m, (v_l^c)_m, (v_u^x)_1, (v_l^x)_1, (v_u^x)_2, (v_l^x)_2, \dots, (v_u^x)_n, (v_l^x)_n$$

in the returned task.

If `NAME_CON` (`NAME_VAR`) is the name of the i th constraint (variable) then the new variables are named as follows:

- The variable corresponding to $(v_u^c)_i$ ($(v_u^x)_i$) is named “`NAME_CON*up`” (“`NAME_VAR*up`”).
- The variable corresponding to $(v_l^c)_i$ ($(v_l^x)_i$) is named “`NAME_CON*lo`” (“`NAME_VAR*lo`”).

where “`*`” can be replaced by a user-defined string by setting the `mosek.sparam.feasrepair_name_separator` parameter.

Please note that if $u_i^c < l_i^c$ or $u_i^x < l_i^x$ then the feasibility repair problem becomes infeasible. Such trivial conflicts must therefore be removed manually before using `Task.relaxprimal`.

The above discussion shows how the function works for a linear optimization problem. However, the function also works for quadratic and conic optimization problems but it cannot be used for general nonlinear optimization problems.

See also:

```
mosek.dparam.feasrepair_tol
mosek.iparam.feasrepair_optimize
mosek.sparam.feasrepair_name_separator
mosek.sparam.feasrepair_name_prefix
```

- `mosek.Task.remove`

Syntax:

```
remove (
    mosek.accmode accmode,
    array(int) sub)
```

accmode (input) Defines if operations are performed row-wise (constraint-oriented) or column-wise (variable-oriented).

sub (input) Indexes of constraints or variables which should be removed.

Description: The function removes a number of constraints or variables from the optimization task. This implies that the existing constraints and variables are renumbered, for instance if constraint 5 is removed then constraint 6 becomes constraint 5 and so forth.

See also:

`Task.append` Appends a number of variables or constraints to the optimization task.

- `mosek.Task.removecone`

Syntax:

```
removecone (int k)
```

k (input) Index of the conic constraint that should be removed.

Description: Removes a conic constraint from the problem. This implies that all the conic constraints appearing after cone number `k` are renumbered, decreasing their indexes by one.

In general, it is much more efficient to remove a cone with a high index than a low index.

- `mosek.Task.resizetask`

Syntax:

```
resizetask (
    int maxnumcon,
    int maxnumvar,
    int maxnumcone,
    int maxnumanz,
    int maxnumqnz)
```

`maxnumcon` (**input**) New maximum number of constraints.

`maxnumvar` (**input**) New maximum number of variables.

`maxnumcone` (**input**) New maximum number of cones.

`maxnumanz` (**input**) New maximum number of non-zeros in A .

`maxnumqnz` (**input**) New maximum number of non-zeros in all Q matrices.

Description: 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 its only function is to give a hint of 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.

See also:

`Task.putmaxnumvar` Sets the number of preallocated variables in the optimization task.

`Task.putmaxnumcon` Sets the number of preallocated constraints in the optimization task.

`Task.putmaxnumcone` Sets the number of preallocated conic constraints in the optimization task.

- `mosek.Task.sensitivityreport`

Syntax:

```
sensitivityreport (mosek.streamtype whichstream)
whichstream (input) Index of the stream.
```

Description: Reads a sensitivity format file from a location given by `mosek.sparam.sensitivity_file_name` and writes the result to the stream `whichstream`. If `mosek.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.

See also:

`Task.dualsensitivity` Performs sensitivity analysis on objective coefficients.

`Task.primalsensitivity` Perform sensitivity analysis on bounds.

`mosek.iparam.log_sensitivity`

`mosek.iparam.log_sensitivity_opt`

`mosek.iparam.sensitivity_type`

- `mosek.Task.set_Stream`

Syntax:

```
set_Stream (mosek.streamtype whichstream)
whichstream Index of the stream.
```

Description: Attach a stream call-back handler.

- `mosek.Task.setdefaults`

Syntax:

```
setdefaults ()
```

Description: Resets all the parameters to their default values.

- `mosek.Task.solutiondef`

Syntax:

```
isdef = solutiondef (mosek.soltype whichsol)
whichsol (input) Selects a solution.
isdef (output) Is non-zero if the requested solution is defined.
```

Description: Checks whether a solution is defined.

- `mosek.Task.solutionsummary`

Syntax:

```
solutionsummary (mosek.streamtype whichstream)
whichstream (input) Index of the stream.
```

Description: Prints a short summary of the current solutions. Please see Section 8.7 for more details.

- `mosek.Task.solvewithbasis`

Syntax:

```
numnz = solvewithbasis (
    int transp,
    int numnz,
    array(int) sub,
    array(double) val)
```

transp (**input**) If this argument is non-zero, then (14.35) is solved. Otherwise the system (14.34) is solved.

numnz (**input/output**) As input it is the number of non-zeros in b . As output it is the number of non-zeros in \bar{x} .

sub (**input/output**) As input it contains the positions of the non-zeros in b , i.e.

$$b[\text{sub}[k]] \neq 0, \quad k = 0, \dots, \text{numnz}[0] - 1.$$

As output it contains the positions of the non-zeros in \bar{x} . It is important that **sub** has room for **numcon** elements.

val (input/output) As input it is the vector b . Although the positions of the non-zero elements are specified in **sub** it is required that $\text{val}[i] = 0$ if $b[i] = 0$. As output **val** is the vector \bar{x} .

Please note that **val** is a dense vector — not a packed sparse vector. This implies that **val** has room for **numcon** elements.

Description: 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

$$B\bar{x} = b \quad (14.34)$$

or the system

$$B^T \bar{x} = b \quad (14.35)$$

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

$$B_{i,k} = \begin{cases} -1, & i = j, \\ 0, & i \neq j. \end{cases}$$

Given the knowledge of how B is constructed it is possible to interpret the solution \bar{x} correctly.

Please note that this function exploits the sparsity in the vector b to speed up the computations.

See also:

Task.initbasissolve Prepare a task for basis solver.
mosek.iparam.basis.solve.use_plus_one

- **mosek.Task.strtoconetype**

Syntax:

conetype = **strtoconetype** (**str** **str**)

str (**input**) String corresponding to the cone type code **codetype**.

conetype (**output**) The cone type corresponding to the string **str**.

Description: Obtains cone type code corresponding to a cone type string.

- **mosek.Task.strtosk**

Syntax:

```
sk = strtosk (str str)
```

str (input) Status key string.

sk (output) Status key corresponding to the string.

Description: Obtains the status key corresponding to an explanatory string.

- `mosek.Task.undefsolution`

Syntax:

```
undefsolution (mosek.soltype whichsol)
```

whichsol (input) Selects a solution.

Description: Undefined a solution. Purges all information regarding `whichsol`.

- `mosek.Task.writebranchpriorities`

Syntax:

```
writebranchpriorities (str filename)
```

filename (input) Data is written to the file `filename`.

Description: Writes branching priority data to a file.

See also:

`Task.readbranchpriorities` Reads branching priority data from a file.

- `mosek.Task.writedata`

Syntax:

```
writedata (str filename)
```

filename (input) Data is written to the file `filename` if it is a nonempty string. Otherwise data is written to the file specified by `mosek.sparam.data_file_name`.

Description: Writes problem data associated with the optimization task to a file in one of four formats:

LP : A text based row oriented format. File extension `.lp`. See Appendix C.

MPS : A text based column oriented format. File extension `.mps`. See Appendix B.

OPF : A text based row oriented format. File extension `.opf`. Supports more problem types than MPS and LP. See Appendix D.

MBT : A binary format for fast reading and writing. File extension `.mbt`.

By default the data file format is determined by the file name extension. This behaviour can be overridden by setting the `mosek.iparam.write_data_format` parameter.

MOSEK is able to read and write files in a compressed format (gzip). To write in the 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 `mosek.iparam.write_generic_names` parameter to `mosek.onoffkey.on`.

See also:

`Task.readdata` Reads problem data from a file.
`mosek.iparam.write_data_format`

- `mosek.Task.writeparamfile`

Syntax:

```
writeparamfile (str filename)
filename (input) The name of parameter file.
```

Description: Writes all the parameters to a parameter file.

- `mosek.Task.writesolution`

Syntax:

```
writesolution (
    mosek.soltype whichsol,
    str filename)
whichsol (input) Selects a solution.
filename (input) A valid file name.
```

Description: Saves the current basic, interior-point, or integer solution to a file.

14.7 Class mosek.Warning

Derived from:

`mosek.Exception`

Description:

This is an exception class representing MOSEK warnings.

14.7.1 Constructors

- `mosek.Warning`

Syntax:

```
Warning (mosek.rescode code)
Construct a warning from a MOSEK error code.
```

DescriptionArguments:

`code` The MOSEK response code to create the exception from.

Chapter 15

Parameter reference

15.1 Parameter groups

Parameters grouped by meaning and functionality.

15.1.1 Logging parameters.

- `mosek.iparam.log` 328
Controls the amount of log information.
- `mosek.iparam.log_bi` 329
Controls the amount of output printed by the basis identification procedure. A higher level implies that more information is logged.
- `mosek.iparam.log_bi_freq` 329
Controls the logging frequency.
- `mosek.iparam.log_concurrent` 329
Controls amount of output printed by the concurrent optimizer.
- `mosek.iparam.log_cut_second_opt` 330
Controls the reduction in the log levels for the second and any subsequent optimizations.
- `mosek.iparam.log_factor` 330
If turned on, then the factor log lines are added to the log.
- `mosek.iparam.log_feasrepair` 330
Controls the amount of output printed when performing feasibility repair.
- `mosek.iparam.log_file` 331
If turned on, then some log info is printed when a file is written or read.

- `mosek.iparam.log_head` 331
If turned on, then a header line is added to the log.
- `mosek.iparam.log_infeas_ana` 331
Controls log level for the infeasibility analyzer.
- `mosek.iparam.log_intpnt` 331
Controls the amount of log information from the interior-point optimizers.
- `mosek.iparam.log_mio` 332
Controls the amount of log information from the mixed-integer optimizers.
- `mosek.iparam.log_mio_freq` 332
The mixed-integer solver logging frequency.
- `mosek.iparam.log_nonconvex` 332
Controls amount of output printed by the nonconvex optimizer.
- `mosek.iparam.log_optimizer` 332
Controls the amount of general optimizer information that is logged.
- `mosek.iparam.log_order` 333
If turned on, then factor lines are added to the log.
- `mosek.iparam.log_param` 333
Controls the amount of information printed out about parameter changes.
- `mosek.iparam.log_presolve` 333
Controls amount of output printed by the presolve procedure. A higher level implies that more information is logged.
- `mosek.iparam.log_response` 333
Controls amount of output printed when response codes are reported. A higher level implies that more information is logged.
- `mosek.iparam.log_sensitivity` 334
Control logging in sensitivity analyzer.
- `mosek.iparam.log_sensitivity_opt` 334
Control logging in sensitivity analyzer.
- `mosek.iparam.log_sim` 334
Controls the amount of log information from the simplex optimizers.
- `mosek.iparam.log_sim_freq` 335
Controls simplex logging frequency.
- `mosek.iparam.log_sim_network_freq` 335
Controls the network simplex logging frequency.
- `mosek.iparam.log_storage` 335
Controls the memory related log information.

15.1.2 Basis identification parameters.

- `mosek.iparam.bi_clean_optimizer` 315
Controls which simplex optimizer is used in the clean-up phase.
- `mosek.iparam.bi_ignore_max_iter` 316
Turns on basis identification in case the interior-point optimizer is terminated due to maximum number of iterations.
- `mosek.iparam.bi_ignore_num_error` 316
Turns on basis identification in case the interior-point optimizer is terminated due to a numerical problem.
- `mosek.iparam.bi_max_iterations` 316
Maximum number of iterations after basis identification.
- `mosek.iparam.intpnt_basis` 321
Controls whether basis identification is performed.
- `mosek.iparam.log_bi` 329
Controls the amount of output printed by the basis identification procedure. A higher level implies that more information is logged.
- `mosek.iparam.log_bi_freq` 329
Controls the logging frequency.
- `mosek.dparam.sim_lu_tol_rel_piv` 300
Relative pivot tolerance employed when computing the LU factorization of the basis matrix.

15.1.3 The Interior-point method parameters.

Parameters defining the behavior of the interior-point method for linear, conic and convex problems.

- `mosek.iparam.bi_ignore_max_iter` 316
Turns on basis identification in case the interior-point optimizer is terminated due to maximum number of iterations.
- `mosek.iparam.bi_ignore_num_error` 316
Turns on basis identification in case the interior-point optimizer is terminated due to a numerical problem.
- `mosek.dparam.check_convexity_rel_tol` 284
Convexity check tolerance.
- `mosek.iparam.intpnt_basis` 321
Controls whether basis identification is performed.
- `mosek.dparam.intpnt_co_tol_dfeas` 287
Dual feasibility tolerance used by the conic interior-point optimizer.

• <code>mosek.dparam.intpnt_co_tol_infeas</code>	287
Infeasibility tolerance for the conic solver.	
• <code>mosek.dparam.intpnt_co_tol_mu_red</code>	287
Optimality tolerance for the conic solver.	
• <code>mosek.dparam.intpnt_co_tol_near_rel</code>	288
Optimality tolerance for the conic solver.	
• <code>mosek.dparam.intpnt_co_tol_pfeas</code>	288
Primal feasibility tolerance used by the conic interior-point optimizer.	
• <code>mosek.dparam.intpnt_co_tol_rel_gap</code>	288
Relative gap termination tolerance used by the conic interior-point optimizer.	
• <code>mosek.iparam.intpnt_diff_step</code>	322
Controls whether different step sizes are allowed in the primal and dual space.	
• <code>mosek.iparam.intpnt_max_iterations</code>	323
Controls the maximum number of iterations allowed in the interior-point optimizer.	
• <code>mosek.iparam.intpnt_max_num_cor</code>	323
Maximum number of correction steps.	
• <code>mosek.iparam.intpnt_max_num_refinement_steps</code>	323
Maximum number of steps to be used by the iterative search direction refinement.	
• <code>mosek.dparam.intpnt_nl_merit_bal</code>	289
Controls if the complementarity and infeasibility is converging to zero at about equal rates.	
• <code>mosek.dparam.intpnt_nl_tol_dfeas</code>	289
Dual feasibility tolerance used when a nonlinear model is solved.	
• <code>mosek.dparam.intpnt_nl_tol_mu_red</code>	289
Relative complementarity gap tolerance.	
• <code>mosek.dparam.intpnt_nl_tol_near_rel</code>	289
Nonlinear solver optimality tolerance parameter.	
• <code>mosek.dparam.intpnt_nl_tol_pfeas</code>	290
Primal feasibility tolerance used when a nonlinear model is solved.	
• <code>mosek.dparam.intpnt_nl_tol_rel_gap</code>	290
Relative gap termination tolerance for nonlinear problems.	
• <code>mosek.dparam.intpnt_nl_tol_rel_step</code>	290
Relative step size to the boundary for general nonlinear optimization problems.	
• <code>mosek.iparam.intpnt_off_col_trh</code>	324
Controls the aggressiveness of the offending column detection.	
• <code>mosek.iparam.intpnt_order_method</code>	324
Controls the ordering strategy.	

- `mosek.iparam.intpnt_regularization_use` 325
Controls whether regularization is allowed.
- `mosek.iparam.intpnt_scaling` 325
Controls how the problem is scaled before the interior-point optimizer is used.
- `mosek.iparam.intpnt_solve_form` 325
Controls whether the primal or the dual problem is solved.
- `mosek.iparam.intpnt_starting_point` 326
Starting point used by the interior-point optimizer.
- `mosek.dparam.intpnt_tol_dfeas` 290
Dual feasibility tolerance used for linear and quadratic optimization problems.
- `mosek.dparam.intpnt_tol_dsafe` 291
Controls the interior-point dual starting point.
- `mosek.dparam.intpnt_tol_infeas` 291
Nonlinear solver infeasibility tolerance parameter.
- `mosek.dparam.intpnt_tol_mu_red` 291
Relative complementarity gap tolerance.
- `mosek.dparam.intpnt_tol_path` 291
interior-point centering aggressiveness.
- `mosek.dparam.intpnt_tol_pfeas` 292
Primal feasibility tolerance used for linear and quadratic optimization problems.
- `mosek.dparam.intpnt_tol_psafe` 292
Controls the interior-point primal starting point.
- `mosek.dparam.intpnt_tol_rel_gap` 292
Relative gap termination tolerance.
- `mosek.dparam.intpnt_tol_rel_step` 292
Relative step size to the boundary for linear and quadratic optimization problems.
- `mosek.dparam.intpnt_tol_step_size` 293
If the step size falls below the value of this parameter, then the interior-point optimizer assumes that it is stalled. It it does not not make any progress.
- `mosek.iparam.log_concurrent` 329
Controls amount of output printed by the concurrent optimizer.
- `mosek.iparam.log_intpnt` 331
Controls the amount of log information from the interior-point optimizers.
- `mosek.iparam.log_presolve` 333
Controls amount of output printed by the presolve procedure. A higher level implies that more information is logged.

- `mosek.dparam.qcqr_reformulate_rel_drop_tol` 299
This parameter determines when columns are dropped in incomplete cholesky factorization doing reformulation of quadratic problems.
- `mosek.iparam.qr_separable_reformulation` 350
Determine if Quadratic programming problems should be reformulated to separable form.

15.1.4 Simplex optimizer parameters.

Parameters defining the behavior of the simplex optimizer for linear problems.

- `mosek.dparam.basis_rel_tol_s` 283
Maximum relative dual bound violation allowed in an optimal basic solution.
- `mosek.iparam.basis_solve_use_plus_one` 315
Controls the sign of the columns in the basis matrix corresponding to slack variables.
- `mosek.dparam.basis_tol_s` 283
Maximum absolute dual bound violation in an optimal basic solution.
- `mosek.dparam.basis_tol_x` 283
Maximum absolute primal bound violation allowed in an optimal basic solution.
- `mosek.iparam.log_sim` 334
Controls the amount of log information from the simplex optimizers.
- `mosek.iparam.log_sim_freq` 335
Controls simplex logging frequency.
- `mosek.iparam.log_sim_minor` 335
Currently not in use.
- `mosek.iparam.sensitivity_optimizer` 357
Controls which optimizer is used for optimal partition sensitivity analysis.
- `mosek.iparam.sim_basis_factor_use` 358
Controls whether a (LU) factorization of the basis is used in a hot-start. Forcing a refactorization sometimes improves the stability of the simplex optimizers, but in most cases there is a performance penalty.
- `mosek.iparam.sim_degen` 358
Controls how aggressively degeneration is handled.
- `mosek.iparam.sim_dual_phaseone_method` 359
An experimental feature.
- `mosek.iparam.sim_exploit_dupvec` 360
Controls if the simplex optimizers are allowed to exploit duplicated columns.

- `mosek.iparam.sim_hotstart` 360
Controls the type of hot-start that the simplex optimizer perform.
- `mosek.iparam.sim_integer` 361
An experimental feature.
- `mosek.dparam.sim_lu_tol_rel_piv` 300
Relative pivot tolerance employed when computing the LU factorization of the basis matrix.
- `mosek.iparam.sim_max_iterations` 361
Maximum number of iterations that can be used by a simplex optimizer.
- `mosek.iparam.sim_max_num_setbacks` 361
Controls how many set-backs that are allowed within a simplex optimizer.
- `mosek.iparam.sim_network_detect_method` 362
Controls which type of detection method the network extraction should use.
- `mosek.iparam.sim_non_singular` 363
Controls if the simplex optimizer ensures a non-singular basis, if possible.
- `mosek.iparam.sim_primal_phaseone_method` 363
An experimental feature.
- `mosek.iparam.sim_reformulation` 365
Controls if the simplex optimizers are allowed to reformulate the problem.
- `mosek.iparam.sim_save_lu` 365
Controls if the LU factorization stored should be replaced with the LU factorization corresponding to the initial basis.
- `mosek.iparam.sim_scaling` 365
Controls how much effort is used in scaling the problem before a simplex optimizer is used.
- `mosek.iparam.sim_scaling_method` 366
Controls how the problem is scaled before a simplex optimizer is used.
- `mosek.iparam.sim_solve_form` 366
Controls whether the primal or the dual problem is solved by the primal-/dual- simplex optimizer.
- `mosek.iparam.sim_stability_priority` 366
Controls how high priority the numerical stability should be given.
- `mosek.iparam.sim_switch_optimizer` 366
Controls the simplex behavior.
- `mosek.dparam.simplex_abs_tol_piv` 300
Absolute pivot tolerance employed by the simplex optimizers.

15.1.5 Primal simplex optimizer parameters.

Parameters defining the behavior of the primal simplex optimizer for linear problems.

- `mosek.iparam.sim_primal_crash` 363
Controls the simplex crash.
- `mosek.iparam.sim_primal_restrict_selection` 363
Controls how aggressively restricted selection is used.
- `mosek.iparam.sim_primal_selection` 364
Controls the primal simplex strategy.

15.1.6 Dual simplex optimizer parameters.

Parameters defining the behavior of the dual simplex optimizer for linear problems.

- `mosek.iparam.sim_dual_crash` 359
Controls whether crashing is performed in the dual simplex optimizer.
- `mosek.iparam.sim_dual_restrict_selection` 359
Controls how aggressively restricted selection is used.
- `mosek.iparam.sim_dual_selection` 359
Controls the dual simplex strategy.

15.1.7 Network simplex optimizer parameters.

Parameters defining the behavior of the network simplex optimizer for linear problems.

- `mosek.iparam.log_sim_network_freq` 335
Controls the network simplex logging frequency.
- `mosek.iparam.sim_network_detect` 362
Level of aggressiveness of network detection.
- `mosek.iparam.sim_network_detect_hotstart` 362
Level of aggressiveness of network detection in a simplex hot-start.
- `mosek.iparam.sim_refactor_freq` 364
Controls the basis refactoring frequency.

15.1.8 Nonlinear convex method parameters.

Parameters defining the behavior of the interior-point method for nonlinear convex problems.

- `mosek.iparam.check_convexity` 317
Specify the level of convexity check on quadratic problems
- `mosek.dparam.intpnt_nl_merit_bal` 289
Controls if the complementarity and infeasibility is converging to zero at about equal rates.
- `mosek.dparam.intpnt_nl_tol_dfeas` 289
Dual feasibility tolerance used when a nonlinear model is solved.
- `mosek.dparam.intpnt_nl_tol_mu_red` 289
Relative complementarity gap tolerance.
- `mosek.dparam.intpnt_nl_tol_near_rel` 289
Nonlinear solver optimality tolerance parameter.
- `mosek.dparam.intpnt_nl_tol_pfeas` 290
Primal feasibility tolerance used when a nonlinear model is solved.
- `mosek.dparam.intpnt_nl_tol_rel_gap` 290
Relative gap termination tolerance for nonlinear problems.
- `mosek.dparam.intpnt_nl_tol_rel_step` 290
Relative step size to the boundary for general nonlinear optimization problems.
- `mosek.dparam.intpnt_tol_infeas` 291
Nonlinear solver infeasibility tolerance parameter.
- `mosek.iparam.log_check_convexity` 329
Controls logging in convexity check on quadratic problems. Set to a positive value to turn logging on.

If a quadratic coefficient matrix is found to violate the requirement of PSD (NSD) then a list of negative (positive) pivot elements is printed. The absolute value of the pivot elements is also shown.

15.1.9 The conic interior-point method parameters.

Parameters defining the behavior of the interior-point method for conic problems.

- `mosek.dparam.intpnt_co_tol_dfeas` 287
Dual feasibility tolerance used by the conic interior-point optimizer.
- `mosek.dparam.intpnt_co_tol_infeas` 287
Infeasibility tolerance for the conic solver.
- `mosek.dparam.intpnt_co_tol_mu_red` 287
Optimality tolerance for the conic solver.
- `mosek.dparam.intpnt_co_tol_near_rel` 288
Optimality tolerance for the conic solver.

- `mosek.dparam.intpnt_co_tol_pfeas` 288
Primal feasibility tolerance used by the conic interior-point optimizer.
- `mosek.dparam.intpnt_co_tol_rel_gap` 288
Relative gap termination tolerance used by the conic interior-point optimizer.

15.1.10 The mixed-integer optimization parameters.

- `mosek.iparam.log_mio` 332
Controls the amount of log information from the mixed-integer optimizers.
- `mosek.iparam.log_mio_freq` 332
The mixed-integer solver logging frequency.
- `mosek.iparam.mio_branch_dir` 336
Controls whether the mixed-integer optimizer is branching up or down by default.
- `mosek.iparam.mio_branch_priorities_use` 336
Controls whether branching priorities are used by the mixed-integer optimizer.
- `mosek.iparam.mio_construct_sol` 337
Controls if an initial mixed integer solution should be constructed from the values of the integer variables.
- `mosek.iparam.mio_cont_sol` 337
Controls the meaning of interior-point and basic solutions in mixed integer problems.
- `mosek.iparam.mio_cut_level_root` 337
Controls the cut level employed by the mixed-integer optimizer at the root node.
- `mosek.iparam.mio_cut_level_tree` 338
Controls the cut level employed by the mixed-integer optimizer in the tree.
- `mosek.dparam.mio_disable_term_time` 293
Certain termination criteria is disabled within the mixed-integer optimizer for period time specified by the parameter.
- `mosek.iparam.mio_feaspump_level` 338
Controls the feasibility pump heuristic which is used to construct a good initial feasible solution.
- `mosek.iparam.mio_heuristic_level` 339
Controls the heuristic employed by the mixed-integer optimizer to locate an initial integer feasible solution.
- `mosek.dparam.mio_heuristic_time` 294
Time limit for the mixed-integer heuristics.
- `mosek.iparam.mio_hotstart` 339
Controls whether the integer optimizer is hot-started.

- `mosek.iparam.mio_keep_basis` 339
Controls whether the integer presolve keeps bases in memory.
- `mosek.iparam.mio_max_num_branches` 340
Maximum number of branches allowed during the branch and bound search.
- `mosek.iparam.mio_max_num_relaxs` 340
Maximum number of relaxations in branch and bound search.
- `mosek.iparam.mio_max_num_solutions` 340
Controls how many feasible solutions the mixed-integer optimizer investigates.
- `mosek.dparam.mio_max_time` 294
Time limit for the mixed-integer optimizer.
- `mosek.dparam.mio_max_time_aprx_opt` 295
Time limit for the mixed-integer optimizer.
- `mosek.dparam.mio_near_tol_abs_gap` 295
Relaxed absolute optimality tolerance employed by the mixed-integer optimizer.
- `mosek.dparam.mio_near_tol_rel_gap` 295
The mixed-integer optimizer is terminated when this tolerance is satisfied.
- `mosek.iparam.mio_node_optimizer` 341
Controls which optimizer is employed at the non-root nodes in the mixed-integer optimizer.
- `mosek.iparam.mio_node_selection` 342
Controls the node selection strategy employed by the mixed-integer optimizer.
- `mosek.iparam.mio_optimizer_mode` 342
An experimental feature.
- `mosek.iparam.mio_presolve_aggregate` 342
Controls whether problem aggregation is performed in the mixed-integer presolve.
- `mosek.iparam.mio_presolve_probing` 343
Controls whether probing is employed by the mixed-integer presolve.
- `mosek.iparam.mio_presolve_use` 343
Controls whether presolve is performed by the mixed-integer optimizer.
- `mosek.dparam.mio_rel_add_cut_limited` 296
Controls cut generation for mixed-integer optimizer.
- `mosek.dparam.mio_rel_gap_const` 296
This value is used to compute the relative gap for the solution to an integer optimization problem.
- `mosek.iparam.mio_root_optimizer` 343
Controls which optimizer is employed at the root node in the mixed-integer optimizer.
- `mosek.iparam.mio_strong_branch` 344
The depth from the root in which strong branching is employed.

- `mosek.dparam.mio_tol_abs_gap` 296
Absolute optimality tolerance employed by the mixed-integer optimizer.
- `mosek.dparam.mio_tol_abs_relax_int` 296
Integer constraint tolerance.
- `mosek.dparam.mio_tol_feas` 297
Feasibility tolerance for mixed integer solver. Any solution with maximum infeasibility below this value will be considered feasible.
- `mosek.dparam.mio_tol_rel_gap` 297
Relative optimality tolerance employed by the mixed-integer optimizer.
- `mosek.dparam.mio_tol_rel_relax_int` 297
Integer constraint tolerance.
- `mosek.dparam.mio_tol_x` 297
Absolute solution tolerance used in mixed-integer optimizer.

15.1.11 Presolve parameters.

- `mosek.iparam.presolve_elim_fill` 348
Maximum amount of fill-in in the elimination phase.
- `mosek.iparam.presolve_eliminator_max_num_tries` 348
Control the maximum number of times the eliminator is tried.
- `mosek.iparam.presolve_eliminator_use` 348
Controls whether free or implied free variables are eliminated from the problem.
- `mosek.iparam.presolve_level` 349
Currently not used.
- `mosek.iparam.presolve_lindep_use` 349
Controls whether the linear constraints are checked for linear dependencies.
- `mosek.iparam.presolve_lindep_work_lim` 349
Controls linear dependency check in presolve.
- `mosek.dparam.presolve_tol_aij` 298
Absolute zero tolerance employed for constraint coefficients in the presolve.
- `mosek.dparam.presolve_tol_lin_dep` 299
Controls when a constraint is determined to be linearly dependent.
- `mosek.dparam.presolve_tol_s` 299
Absolute zero tolerance employed for slack variables in the presolve.
- `mosek.dparam.presolve_tol_x` 299
Absolute zero tolerance employed for variables in the presolve.
- `mosek.iparam.presolve_use` 350
Controls whether the presolve is applied to a problem before it is optimized.

15.1.12 Termination criterion parameters.

Parameters which define termination and optimality criteria and related information.

- `mosek.dparam.basis_rel_tol_s` 283
Maximum relative dual bound violation allowed in an optimal basic solution.
- `mosek.dparam.basis_tol_s` 283
Maximum absolute dual bound violation in an optimal basic solution.
- `mosek.dparam.basis_tol_x` 283
Maximum absolute primal bound violation allowed in an optimal basic solution.
- `mosek.iparam.bi_max_iterations` 316
Maximum number of iterations after basis identification.
- `mosek.dparam.intpnt_co_tol_dfeas` 287
Dual feasibility tolerance used by the conic interior-point optimizer.
- `mosek.dparam.intpnt_co_tol_infeas` 287
Infeasibility tolerance for the conic solver.
- `mosek.dparam.intpnt_co_tol_mu_red` 287
Optimality tolerance for the conic solver.
- `mosek.dparam.intpnt_co_tol_near_rel` 288
Optimality tolerance for the conic solver.
- `mosek.dparam.intpnt_co_tol_pfeas` 288
Primal feasibility tolerance used by the conic interior-point optimizer.
- `mosek.dparam.intpnt_co_tol_rel_gap` 288
Relative gap termination tolerance used by the conic interior-point optimizer.
- `mosek.iparam.intpnt_max_iterations` 323
Controls the maximum number of iterations allowed in the interior-point optimizer.
- `mosek.dparam.intpnt_nl_tol_dfeas` 289
Dual feasibility tolerance used when a nonlinear model is solved.
- `mosek.dparam.intpnt_nl_tol_mu_red` 289
Relative complementarity gap tolerance.
- `mosek.dparam.intpnt_nl_tol_near_rel` 289
Nonlinear solver optimality tolerance parameter.
- `mosek.dparam.intpnt_nl_tol_pfeas` 290
Primal feasibility tolerance used when a nonlinear model is solved.
- `mosek.dparam.intpnt_nl_tol_rel_gap` 290
Relative gap termination tolerance for nonlinear problems.

• <code>mosek.dparam.intpnt_tol_dfeas</code>	290
Dual feasibility tolerance used for linear and quadratic optimization problems.	
• <code>mosek.dparam.intpnt_tol_infeas</code>	291
Nonlinear solver infeasibility tolerance parameter.	
• <code>mosek.dparam.intpnt_tol_mu_red</code>	291
Relative complementarity gap tolerance.	
• <code>mosek.dparam.intpnt_tol_pfeas</code>	292
Primal feasibility tolerance used for linear and quadratic optimization problems.	
• <code>mosek.dparam.intpnt_tol_rel_gap</code>	292
Relative gap termination tolerance.	
• <code>mosek.dparam.lower_obj_cut</code>	293
Objective bound.	
• <code>mosek.dparam.lower_obj_cut_finite_trh</code>	293
Objective bound.	
• <code>mosek.dparam.mio_disable_term_time</code>	293
Certain termination criteria is disabled within the mixed-integer optimizer for period time specified by the parameter.	
• <code>mosek.iparam.mio_max_num_branches</code>	340
Maximum number of branches allowed during the branch and bound search.	
• <code>mosek.iparam.mio_max_num_solutions</code>	340
Controls how many feasible solutions the mixed-integer optimizer investigates.	
• <code>mosek.dparam.mio_max_time</code>	294
Time limit for the mixed-integer optimizer.	
• <code>mosek.dparam.mio_near_tol_rel_gap</code>	295
The mixed-integer optimizer is terminated when this tolerance is satisfied.	
• <code>mosek.dparam.mio_rel_gap_const</code>	296
This value is used to compute the relative gap for the solution to an integer optimization problem.	
• <code>mosek.dparam.mio_tol_rel_gap</code>	297
Relative optimality tolerance employed by the mixed-integer optimizer.	
• <code>mosek.dparam.optimizer_max_time</code>	298
Solver time limit.	
• <code>mosek.iparam.sim_max_iterations</code>	361
Maximum number of iterations that can be used by a simplex optimizer.	
• <code>mosek.dparam.upper_obj_cut</code>	300
Objective bound.	
• <code>mosek.dparam.upper_obj_cut_finite_trh</code>	300
Objective bound.	

15.1.13 Progress call-back parameters.

- `mosek.dparam.callback_freq` 283
Controls progress call-back frequency.
- `mosek.iparam.solution_callback` 368
Indicates whether solution call-backs will be performed during the optimization.

15.1.14 Non-convex solver parameters.

- `mosek.iparam.log_nonconvex` 332
Controls amount of output printed by the nonconvex optimizer.
- `mosek.iparam.nonconvex_max_iterations` 344
Maximum number of iterations that can be used by the nonconvex optimizer.
- `mosek.dparam.nonconvex_tol_feas` 298
Feasibility tolerance used by the nonconvex optimizer.
- `mosek.dparam.nonconvex_tol_opt` 298
Optimality tolerance used by the nonconvex optimizer.

15.1.15 Feasibility repair parameters.

- `mosek.dparam.feasrepair_tol` 287
Tolerance for constraint enforcing upper bound on sum of weighted violations in feasibility repair.

15.1.16 Optimization system parameters.

Parameters defining the overall solver system environment. This includes system and platform related information and behavior.

- `mosek.iparam.auto_update_sol_info` 314
Controls whether the solution information items are automatically updated after an optimization is performed.
- `mosek.iparam.cache_license` 316
Control license caching.
- `mosek.iparam.cache_size_l1` 317
Specifies the size of the level 1 cache of the processor.
- `mosek.iparam.cache_size_l2` 317
Specifies the size of the level 2 cache of the processor.
- `mosek.iparam.cpu_type` 319
Specifies the CPU type.

- `mosek.iparam.intpnt_num_threads` 324
Controls the number of threads employed by the interior-point optimizer. If set to a positive number MOSEK will use this number of threads. If zero the number of threads used will equal the number of cores detected on the machine.
- `mosek.iparam.license_cache_time` 326
Setting this parameter no longer has any effect.
- `mosek.iparam.license_check_time` 327
Controls the license manager client behavior.
- `mosek.iparam.license_wait` 328
Controls if MOSEK should queue for a license if none is available.
- `mosek.iparam.log_storage` 335
Controls the memory related log information.
- `mosek.iparam.timing_level` 369
Controls the a amount of timing performed inside MOSEK.

15.1.17 Output information parameters.

- `mosek.iparam.infeas_report_level` 321
Controls the contents of the infeasibility report.
- `mosek.iparam.license_suppress_expire_wrns` 328
Controls license manager client behavior.
- `mosek.iparam.log` 328
Controls the amount of log information.
- `mosek.iparam.log_bi` 329
Controls the amount of output printed by the basis identification procedure. A higher level implies that more information is logged.
- `mosek.iparam.log_bi_freq` 329
Controls the logging frequency.
- `mosek.iparam.log_cut_second_opt` 330
Controls the reduction in the log levels for the second and any subsequent optimizations.
- `mosek.iparam.log_factor` 330
If turned on, then the factor log lines are added to the log.
- `mosek.iparam.log_feasrepair` 330
Controls the amount of output printed when performing feasibility repair.
- `mosek.iparam.log_file` 331
If turned on, then some log info is printed when a file is written or read.

- `mosek.iparam.log_head` 331
If turned on, then a header line is added to the log.
- `mosek.iparam.log_infeas_ana` 331
Controls log level for the infeasibility analyzer.
- `mosek.iparam.log_intpnt` 331
Controls the amount of log information from the interior-point optimizers.
- `mosek.iparam.log_mio` 332
Controls the amount of log information from the mixed-integer optimizers.
- `mosek.iparam.log_mio_freq` 332
The mixed-integer solver logging frequency.
- `mosek.iparam.log_nonconvex` 332
Controls amount of output printed by the nonconvex optimizer.
- `mosek.iparam.log_optimizer` 332
Controls the amount of general optimizer information that is logged.
- `mosek.iparam.log_order` 333
If turned on, then factor lines are added to the log.
- `mosek.iparam.log_param` 333
Controls the amount of information printed out about parameter changes.
- `mosek.iparam.log_response` 333
Controls amount of output printed when response codes are reported. A higher level implies that more information is logged.
- `mosek.iparam.log_sensitivity` 334
Control logging in sensitivity analyzer.
- `mosek.iparam.log_sensitivity_opt` 334
Control logging in sensitivity analyzer.
- `mosek.iparam.log_sim` 334
Controls the amount of log information from the simplex optimizers.
- `mosek.iparam.log_sim_freq` 335
Controls simplex logging frequency.
- `mosek.iparam.log_sim_minor` 335
Currently not in use.
- `mosek.iparam.log_sim_network_freq` 335
Controls the network simplex logging frequency.
- `mosek.iparam.log_storage` 335
Controls the memory related log information.

- `mosek.iparam.max_num_warnings` 336
Warning level. A higher value results in more warnings.
- `mosek.iparam.warning_level` 369
Warning level.

15.1.18 Extra information about the optimization problem.

- `mosek.iparam.objective_sense` 344
If the objective sense for the task is undefined, then the value of this parameter is used as the default objective sense.

15.1.19 Overall solver parameters.

- `mosek.iparam.bi_clean_optimizer` 315
Controls which simplex optimizer is used in the clean-up phase.
- `mosek.iparam.concurrent_num_optimizers` 318
The maximum number of simultaneous optimizations that will be started by the concurrent optimizer.
- `mosek.iparam.concurrent_priority_dual_simplex` 318
Priority of the dual simplex algorithm when selecting solvers for concurrent optimization.
- `mosek.iparam.concurrent_priority_free_simplex` 318
Priority of the free simplex optimizer when selecting solvers for concurrent optimization.
- `mosek.iparam.concurrent_priority_intpnt` 319
Priority of the interior-point algorithm when selecting solvers for concurrent optimization.
- `mosek.iparam.concurrent_priority_primal_simplex` 319
Priority of the primal simplex algorithm when selecting solvers for concurrent optimization.
- `mosek.iparam.data_check` 320
Enable data checking for debug purposes.
- `mosek.iparam.feasrepair_optimize` 320
Controls which type of feasibility analysis is to be performed.
- `mosek.iparam.infeas_prefer_primal` 321
Controls which certificate is used if both primal- and dual- certificate of infeasibility is available.
- `mosek.iparam.license_wait` 328
Controls if MOSEK should queue for a license if none is available.
- `mosek.iparam.mio_cont_sol` 337
Controls the meaning of interior-point and basic solutions in mixed integer problems.
- `mosek.iparam.mio_local_branch_number` 339
Controls the size of the local search space when doing local branching.

• <code>mosek.iparam.mio_mode</code>	341
Turns on/off the mixed-integer mode.	
• <code>mosek.iparam.optimizer</code>	347
Controls which optimizer is used to optimize the task.	
• <code>mosek.iparam.presolve_level</code>	349
Currently not used.	
• <code>mosek.iparam.presolve_use</code>	350
Controls whether the presolve is applied to a problem before it is optimized.	
• <code>mosek.iparam.sensitivity_all</code>	357
Controls sensitivity report behavior.	
• <code>mosek.iparam.sensitivity_optimizer</code>	357
Controls which optimizer is used for optimal partition sensitivity analysis.	
• <code>mosek.iparam.sensitivity_type</code>	358
Controls which type of sensitivity analysis is to be performed.	
• <code>mosek.iparam.solution_callback</code>	368
Indicates whether solution call-backs will be performed during the optimization.	

15.1.20 Behavior of the optimization task.

Parameters defining the behavior of an optimization task when loading data.

• <code>mosek.iparam.alloc_add_qnz</code>	313
Controls how the quadratic matrixes are extended.	
• <code>mosek.sparam.feasrepair_name_prefix</code>	379
Feasibility repair name prefix.	
• <code>mosek.sparam.feasrepair_name_separator</code>	379
Feasibility repair name separator.	
• <code>mosek.sparam.feasrepair_name_wsumviol</code>	379
Feasibility repair name violation name.	
• <code>mosek.iparam.read_add_anz</code>	350
Controls how the constraint matrix is extended.	
• <code>mosek.iparam.read_add_con</code>	350
Additional number of constraints that is made room for in the problem.	
• <code>mosek.iparam.read_add_cone</code>	351
Additional number of conic constraints that is made room for in the problem.	
• <code>mosek.iparam.read_add_qnz</code>	351
Controls how the quadratic matrixes are extended.	

- `mosek.iparam.read_add_var` 351
Additional number of variables that is made room for in the problem.
- `mosek.iparam.read_anz` 351
Controls the expected number of constraint non-zeros.
- `mosek.iparam.read_con` 352
Controls the expected number of constraints.
- `mosek.iparam.read_cone` 352
Controls the expected number of conic constraints.
- `mosek.iparam.read_qnz` 356
Controls the expected number of quadratic non-zeros.
- `mosek.iparam.read_task_ignore_param` 356
Controls what information is used from the task files.
- `mosek.iparam.read_var` 356
Controls the expected number of variables.
- `mosek.iparam.write_task_inc_sol` 376
Controls whether the solutions are stored in the task file too.

15.1.21 Data input/output parameters.

Parameters defining the behavior of data readers and writers.

- `mosek.sparam.bas_sol_file_name` 378
Name of the bas solution file.
- `mosek.sparam.data_file_name` 378
Data are read and written to this file.
- `mosek.sparam.debug_file_name` 379
MOSEK debug file.
- `mosek.iparam.infeas_report_auto` 321
Turns the feasibility report on or off.
- `mosek.sparam.int_sol_file_name` 380
Name of the int solution file.
- `mosek.sparam.itr_sol_file_name` 380
Name of the itr solution file.
- `mosek.iparam.log_file` 331
If turned on, then some log info is printed when a file is written or read.
- `mosek.iparam.lp_write_ignore_incompatible_items` 336
Controls the result of writing a problem containing incompatible items to an LP file.

- `mosek.iparam.opf_max_terms_per_line` 344
The maximum number of terms (linear and quadratic) per line when an OPF file is written.
- `mosek.iparam.opf_write_header` 345
Write a text header with date and MOSEK version in an OPF file.
- `mosek.iparam.opf_write_hints` 345
Write a hint section with problem dimensions in the beginning of an OPF file.
- `mosek.iparam.opf_write_parameters` 345
Write a parameter section in an OPF file.
- `mosek.iparam.opf_write_problem` 346
Write objective, constraints, bounds etc. to an OPF file.
- `mosek.iparam.opf_write_sol_bas` 346
Controls what is written to the OPF files.
- `mosek.iparam.opf_write_sol_itg` 346
Controls what is written to the OPF files.
- `mosek.iparam.opf_write_sol_itr` 346
Controls what is written to the OPF files.
- `mosek.iparam.opf_write_solutions` 347
Enable inclusion of solutions in the OPF files.
- `mosek.sparam.param_comment_sign` 380
Solution file comment character.
- `mosek.iparam.param_read_case_name` 347
If turned on, then names in the parameter file are case sensitive.
- `mosek.sparam.param_read_file_name` 380
Modifications to the parameter database is read from this file.
- `mosek.iparam.param_read_ign_error` 348
If turned on, then errors in parameter settings is ignored.
- `mosek.sparam.param_write_file_name` 381
The parameter database is written to this file.
- `mosek.iparam.read_add_anz` 350
Controls how the constraint matrix is extended.
- `mosek.iparam.read_add_con` 350
Additional number of constraints that is made room for in the problem.
- `mosek.iparam.read_add_cone` 351
Additional number of conic constraints that is made room for in the problem.
- `mosek.iparam.read_add_qnz` 351
Controls how the quadratic matrixes are extended.

- `mosek.iparam.read_add_var` 351
Additional number of variables that is made room for in the problem.
- `mosek.iparam.read_anz` 351
Controls the expected number of constraint non-zeros.
- `mosek.iparam.read_con` 352
Controls the expected number of constraints.
- `mosek.iparam.read_cone` 352
Controls the expected number of conic constraints.
- `mosek.iparam.read_data_compressed` 352
Controls the input file decompression.
- `mosek.iparam.read_data_format` 352
Format of the data file to be read.
- `mosek.iparam.read_keep_free_con` 353
Controls whether the free constraints are included in the problem.
- `mosek.iparam.read_lp_drop_new_vars_in_bou` 353
Controls how the LP files are interpreted.
- `mosek.iparam.read_lp_quoted_names` 353
If a name is in quotes when reading an LP file, the quotes will be removed.
- `mosek.sparam.read_mps_bou_name` 381
Name of the BOUNDS vector used. An empty name means that the first BOUNDS vector is used.
- `mosek.iparam.read_mps_format` 354
Controls how strictly the MPS file reader interprets the MPS format.
- `mosek.iparam.read_mps_keep_int` 354
Controls if integer constraints are read.
- `mosek.sparam.read_mps_obj_name` 381
Objective name in the MPS file.
- `mosek.iparam.read_mps_obj_sense` 354
Controls the MPS format extensions.
- `mosek.iparam.read_mps_quoted_names` 355
Controls the MPS format extensions.
- `mosek.sparam.read_mps_ran_name` 381
Name of the RANGE vector used. An empty name means that the first RANGE vector is used.
- `mosek.iparam.read_mps_relax` 355
Controls the meaning of integer constraints.

• <code>mosek.sparam.read_mps_rhs_name</code>	382
Name of the RHS used. An empty name means that the first RHS vector is used.	
• <code>mosek.iparam.read_mps_width</code>	355
Controls the maximal number of characters allowed in one line of the MPS file.	
• <code>mosek.iparam.read_q_mode</code>	355
Controls how the Q matrices are read from the MPS file.	
• <code>mosek.iparam.read_qnz</code>	356
Controls the expected number of quadratic non-zeros.	
• <code>mosek.iparam.read_task_ignore_param</code>	356
Controls what information is used from the task files.	
• <code>mosek.iparam.read_var</code>	356
Controls the expected number of variables.	
• <code>mosek.sparam.sensitivity_file_name</code>	382
Sensitivity report file name.	
• <code>mosek.sparam.sensitivity_res_file_name</code>	382
Name of the sensitivity report output file.	
• <code>mosek.sparam.sol_filter_xc_low</code>	382
Solution file filter.	
• <code>mosek.sparam.sol_filter_xc_upr</code>	383
Solution file filter.	
• <code>mosek.sparam.sol_filter_xx_low</code>	383
Solution file filter.	
• <code>mosek.sparam.sol_filter_xx_upr</code>	383
Solution file filter.	
• <code>mosek.iparam.sol_quoted_names</code>	367
Controls the solution file format.	
• <code>mosek.iparam.sol_read_name_width</code>	368
Controls the input solution file format.	
• <code>mosek.iparam.sol_read_width</code>	368
Controls the input solution file format.	
• <code>mosek.sparam.stat_file_name</code>	384
Statistics file name.	
• <code>mosek.sparam.stat_key</code>	384
Key used when writing the summary file.	
• <code>mosek.sparam.stat_name</code>	384
Name used when writing the statistics file.	

• <code>mosek.iparam.write_bas_constraints</code>	369
Controls the basic solution file format.	
• <code>mosek.iparam.write_bas_head</code>	369
Controls the basic solution file format.	
• <code>mosek.iparam.write_bas_variables</code>	370
Controls the basic solution file format.	
• <code>mosek.iparam.write_data_compressed</code>	370
Controls output file compression.	
• <code>mosek.iparam.write_data_format</code>	370
Controls the output file format.	
• <code>mosek.iparam.write_data_param</code>	371
Controls output file data.	
• <code>mosek.iparam.write_free_con</code>	371
Controls the output file data.	
• <code>mosek.iparam.write_generic_names</code>	371
Controls the output file data.	
• <code>mosek.iparam.write_generic_names_io</code>	371
Index origin used in generic names.	
• <code>mosek.iparam.write_int_constraints</code>	372
Controls the integer solution file format.	
• <code>mosek.iparam.write_int_head</code>	372
Controls the integer solution file format.	
• <code>mosek.iparam.write_int_variables</code>	372
Controls the integer solution file format.	
• <code>mosek.sparam.write_lp_gen_var_name</code>	384
Added variable names in the LP files.	
• <code>mosek.iparam.write_lp_line_width</code>	373
Controls the LP output file format.	
• <code>mosek.iparam.write_lp_quoted_names</code>	373
Controls LP output file format.	
• <code>mosek.iparam.write_lp_strict_format</code>	373
Controls whether LP output files satisfy the LP format strictly.	
• <code>mosek.iparam.write_lp_terms_per_line</code>	373
Controls the LP output file format.	
• <code>mosek.iparam.write_mps_int</code>	374
Controls the output file data.	

- `mosek.iparam.write_mps_obj_sense` 374
Controls the output file data.
- `mosek.iparam.write_mps_quoted_names` 374
Controls the output file data.
- `mosek.iparam.write_mps_strict` 374
Controls the output MPS file format.
- `mosek.iparam.write_precision` 375
Controls data precision employed in when writing an MPS file.
- `mosek.iparam.write_sol_constraints` 375
Controls the solution file format.
- `mosek.iparam.write_sol_head` 375
Controls solution file format.
- `mosek.iparam.write_sol_variables` 376
Controls the solution file format.
- `mosek.iparam.write_task_inc_sol` 376
Controls whether the solutions are stored in the task file too.
- `mosek.iparam.write_xml_mode` 376
Controls if linear coefficients should be written by row or column when writing in the XML file format.

15.1.22 Analysis parameters.

Parameters controlling the behaviour of the problem and solution analyzers.

- `mosek.iparam.ana_sol_basis` 313
Controls whether the basis matrix is analyzed in solution analyzer.
- `mosek.dparam.ana_sol_infeas_tol` 282
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.
- `mosek.iparam.ana_sol_print_violated` 314
Controls whether a list of violated constraints is printed.

15.1.23 Solution input/output parameters.

Parameters defining the behavior of solution reader and writer.

- `mosek.sparam.bas_sol_file_name` 378
Name of the bas solution file.

• <code>mosek.iparam.infeas_report_auto</code>	321
Turns the feasibility report on or off.	
• <code>mosek.sparam.int_sol_file_name</code>	380
Name of the int solution file.	
• <code>mosek.sparam.itr_sol_file_name</code>	380
Name of the itr solution file.	
• <code>mosek.iparam.sol_filter_keep_basic</code>	367
Controls the license manager client behavior.	
• <code>mosek.iparam.sol_filter_keep_ranged</code>	367
Control the contents of the solution files.	
• <code>mosek.sparam.sol_filter_xc_low</code>	382
Solution file filter.	
• <code>mosek.sparam.sol_filter_xc_upr</code>	383
Solution file filter.	
• <code>mosek.sparam.sol_filter_xx_low</code>	383
Solution file filter.	
• <code>mosek.sparam.sol_filter_xx_upr</code>	383
Solution file filter.	
• <code>mosek.iparam.sol_quoted_names</code>	367
Controls the solution file format.	
• <code>mosek.iparam.sol_read_name_width</code>	368
Controls the input solution file format.	
• <code>mosek.iparam.sol_read_width</code>	368
Controls the input solution file format.	
• <code>mosek.iparam.write_bas_constraints</code>	369
Controls the basic solution file format.	
• <code>mosek.iparam.write_bas_head</code>	369
Controls the basic solution file format.	
• <code>mosek.iparam.write_bas_variables</code>	370
Controls the basic solution file format.	
• <code>mosek.iparam.write_int_constraints</code>	372
Controls the integer solution file format.	
• <code>mosek.iparam.write_int_head</code>	372
Controls the integer solution file format.	
• <code>mosek.iparam.write_int_variables</code>	372
Controls the integer solution file format.	

- `mosek.iparam.write_sol_constraints` 375
Controls the solution file format.
- `mosek.iparam.write_sol_head`..... 375
Controls solution file format.
- `mosek.iparam.write_sol_variables` 376
Controls the solution file format.

15.1.24 Infeasibility report parameters.

- `mosek.iparam.infeas_generic_names`..... 320
Controls the contents of the infeasibility report.
- `mosek.iparam.infeas_report_level`..... 321
Controls the contents of the infeasibility report.
- `mosek.iparam.log_infeas_ana`..... 331
Controls log level for the infeasibility analyzer.

15.1.25 License manager parameters.

- `mosek.iparam.license_allow_overuse` 326
Controls if license overuse is allowed when caching licenses
- `mosek.iparam.license_cache_time`..... 326
Setting this parameter no longer has any effect.
- `mosek.iparam.license_check_time`..... 327
Controls the license manager client behavior.
- `mosek.iparam.license_debug`..... 327
Controls the license manager client debugging behavior.
- `mosek.iparam.license_pause_time`..... 327
Controls license manager client behavior.
- `mosek.iparam.license_suppress_expire_wrns` 328
Controls license manager client behavior.
- `mosek.iparam.license_wait`..... 328
Controls if MOSEK should queue for a license if none is available.

15.1.26 Data check parameters.

These parameters defines data checking settings and problem data tolerances, i.e. which values are rounded to 0 or infinity, and which values are large or small enough to produce a warning.

- `mosek.iparam.check_convexity` 317
Specify the level of convexity check on quadratic problems
- `mosek.iparam.check_task_data` 318
If this feature is turned on, then the task data is checked for bad values i.e. NaNs. before an optimization is performed.
- `mosek.dparam.data_tol_aij` 284
Data tolerance threshold.
- `mosek.dparam.data_tol_aij_huge` 284
Data tolerance threshold.
- `mosek.dparam.data_tol_aij_large` 285
Data tolerance threshold.
- `mosek.dparam.data_tol_bound_inf` 285
Data tolerance threshold.
- `mosek.dparam.data_tol_bound_wrn` 285
Data tolerance threshold.
- `mosek.dparam.data_tol_c_huge` 285
Data tolerance threshold.
- `mosek.dparam.data_tol_cj_large` 286
Data tolerance threshold.
- `mosek.dparam.data_tol_qij` 286
Data tolerance threshold.
- `mosek.dparam.data_tol_x` 286
Data tolerance threshold.
- `mosek.iparam.log_check_convexity` 329
Controls logging in convexity check on quadratic problems. Set to a positive value to turn logging on.

If a quadratic coefficient matrix is found to violate the requirement of PSD (NSD) then a list of negative (positive) pivot elements is printed. The absolute value of the pivot elements is also shown.

15.1.27 Debugging parameters.

These parameters defines that can be used when debugging a problem.

- `mosek.iparam.auto_sort_a_before_opt` 314
Controls whether the elements in each column of A are sorted before an optimization is performed.
- `mosek.iparam.check_task_data` 318
If this feature is turned on, then the task data is checked for bad values i.e. NaNs. before an optimization is performed.

15.2 Double parameters

- `mosek.dparam.ana_sol_infeas_tol` 282
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.
- `mosek.dparam.basis_rel_tol_s` 283
Maximum relative dual bound violation allowed in an optimal basic solution.
- `mosek.dparam.basis_tol_s` 283
Maximum absolute dual bound violation in an optimal basic solution.
- `mosek.dparam.basis_tol_x` 283
Maximum absolute primal bound violation allowed in an optimal basic solution.
- `mosek.dparam.callback_freq` 283
Controls progress call-back frequency.
- `mosek.dparam.check_convexity_rel_tol` 284
Convexity check tolerance.
- `mosek.dparam.data_tol_aij` 284
Data tolerance threshold.
- `mosek.dparam.data_tol_aij_huge` 284
Data tolerance threshold.
- `mosek.dparam.data_tol_aij_large` 285
Data tolerance threshold.
- `mosek.dparam.data_tol_bound_inf` 285
Data tolerance threshold.
- `mosek.dparam.data_tol_bound_wrn` 285
Data tolerance threshold.
- `mosek.dparam.data_tol_c_huge` 285
Data tolerance threshold.
- `mosek.dparam.data_tol_cj_large` 286
Data tolerance threshold.
- `mosek.dparam.data_tol_qij` 286
Data tolerance threshold.
- `mosek.dparam.data_tol_x` 286
Data tolerance threshold.
- `mosek.dparam.feasrepair_tol` 287
Tolerance for constraint enforcing upper bound on sum of weighted violations in feasibility repair.

• <code>mosek.dparam.intpnt_co_tol_dfeas</code>	287
Dual feasibility tolerance used by the conic interior-point optimizer.	
• <code>mosek.dparam.intpnt_co_tol_infeas</code>	287
Infeasibility tolerance for the conic solver.	
• <code>mosek.dparam.intpnt_co_tol_mu_red</code>	287
Optimality tolerance for the conic solver.	
• <code>mosek.dparam.intpnt_co_tol_near_rel</code>	288
Optimality tolerance for the conic solver.	
• <code>mosek.dparam.intpnt_co_tol_pfeas</code>	288
Primal feasibility tolerance used by the conic interior-point optimizer.	
• <code>mosek.dparam.intpnt_co_tol_rel_gap</code>	288
Relative gap termination tolerance used by the conic interior-point optimizer.	
• <code>mosek.dparam.intpnt_nl_merit_bal</code>	289
Controls if the complementarity and infeasibility is converging to zero at about equal rates.	
• <code>mosek.dparam.intpnt_nl_tol_dfeas</code>	289
Dual feasibility tolerance used when a nonlinear model is solved.	
• <code>mosek.dparam.intpnt_nl_tol_mu_red</code>	289
Relative complementarity gap tolerance.	
• <code>mosek.dparam.intpnt_nl_tol_near_rel</code>	289
Nonlinear solver optimality tolerance parameter.	
• <code>mosek.dparam.intpnt_nl_tol_pfeas</code>	290
Primal feasibility tolerance used when a nonlinear model is solved.	
• <code>mosek.dparam.intpnt_nl_tol_rel_gap</code>	290
Relative gap termination tolerance for nonlinear problems.	
• <code>mosek.dparam.intpnt_nl_tol_rel_step</code>	290
Relative step size to the boundary for general nonlinear optimization problems.	
• <code>mosek.dparam.intpnt_tol_dfeas</code>	290
Dual feasibility tolerance used for linear and quadratic optimization problems.	
• <code>mosek.dparam.intpnt_tol_dsafe</code>	291
Controls the interior-point dual starting point.	
• <code>mosek.dparam.intpnt_tol_infeas</code>	291
Nonlinear solver infeasibility tolerance parameter.	
• <code>mosek.dparam.intpnt_tol_mu_red</code>	291
Relative complementarity gap tolerance.	
• <code>mosek.dparam.intpnt_tol_path</code>	291
interior-point centering aggressiveness.	

- `mosek.dparam.intpnt_tol_pfeas` 292
Primal feasibility tolerance used for linear and quadratic optimization problems.
- `mosek.dparam.intpnt_tol_psafe` 292
Controls the interior-point primal starting point.
- `mosek.dparam.intpnt_tol_rel_gap` 292
Relative gap termination tolerance.
- `mosek.dparam.intpnt_tol_rel_step` 292
Relative step size to the boundary for linear and quadratic optimization problems.
- `mosek.dparam.intpnt_tol_step_size` 293
If the step size falls below the value of this parameter, then the interior-point optimizer assumes that it is stalled. It it does not not make any progress.
- `mosek.dparam.lower_obj_cut` 293
Objective bound.
- `mosek.dparam.lower_obj_cut_finite_trh` 293
Objective bound.
- `mosek.dparam.mio_disable_term_time` 293
Certain termination criteria is disabled within the mixed-integer optimizer for period time specified by the parameter.
- `mosek.dparam.mio_heuristic_time` 294
Time limit for the mixed-integer heuristics.
- `mosek.dparam.mio_max_time` 294
Time limit for the mixed-integer optimizer.
- `mosek.dparam.mio_max_time_aprx_opt` 295
Time limit for the mixed-integer optimizer.
- `mosek.dparam.mio_near_tol_abs_gap` 295
Relaxed absolute optimality tolerance employed by the mixed-integer optimizer.
- `mosek.dparam.mio_near_tol_rel_gap` 295
The mixed-integer optimizer is terminated when this tolerance is satisfied.
- `mosek.dparam.mio_rel_add_cut_limited` 296
Controls cut generation for mixed-integer optimizer.
- `mosek.dparam.mio_rel_gap_const` 296
This value is used to compute the relative gap for the solution to an integer optimization problem.
- `mosek.dparam.mio_tol_abs_gap` 296
Absolute optimality tolerance employed by the mixed-integer optimizer.
- `mosek.dparam.mio_tol_abs_relax_int` 296
Integer constraint tolerance.

- `mosek.dparam.mio_tol_feas` 297
Feasibility tolerance for mixed integer solver. Any solution with maximum infeasibility below this value will be considered feasible.
- `mosek.dparam.mio_tol_rel_gap` 297
Relative optimality tolerance employed by the mixed-integer optimizer.
- `mosek.dparam.mio_tol_rel_relax_int` 297
Integer constraint tolerance.
- `mosek.dparam.mio_tol_x` 297
Absolute solution tolerance used in mixed-integer optimizer.
- `mosek.dparam.nonconvex_tol_feas` 298
Feasibility tolerance used by the nonconvex optimizer.
- `mosek.dparam.nonconvex_tol_opt` 298
Optimality tolerance used by the nonconvex optimizer.
- `mosek.dparam.optimizer_max_time` 298
Solver time limit.
- `mosek.dparam.presolve_tol_aij` 298
Absolute zero tolerance employed for constraint coefficients in the presolve.
- `mosek.dparam.presolve_tol_lin_dep` 299
Controls when a constraint is determined to be linearly dependent.
- `mosek.dparam.presolve_tol_s` 299
Absolute zero tolerance employed for slack variables in the presolve.
- `mosek.dparam.presolve_tol_x` 299
Absolute zero tolerance employed for variables in the presolve.
- `mosek.dparam.qcqp_reformulate_rel_drop_tol` 299
This parameter determines when columns are dropped in incomplete cholesky factorization doing reformulation of quadratic problems.
- `mosek.dparam.sim_lu_tol_rel_piv` 300
Relative pivot tolerance employed when computing the LU factorization of the basis matrix.
- `mosek.dparam.simplex_abs_tol_piv` 300
Absolute pivot tolerance employed by the simplex optimizers.
- `mosek.dparam.upper_obj_cut` 300
Objective bound.
- `mosek.dparam.upper_obj_cut_finite_trh` 300
Objective bound.
- `ana_sol_infeas_tol`

Corresponding constant:`mosek.dparam.ana_sol_infeas_tol`**Description:**

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.

Possible Values:

Any number between 0.0 and +inf.

Default value:`+1e-8`

- `basis_rel_tol_s`

Corresponding constant:`mosek.dparam.basis_rel_tol_s`**Description:**

Maximum relative dual bound violation allowed in an optimal basic solution.

Possible Values:

Any number between 0.0 and +inf.

Default value:`1.0e-12`

- `basis_tol_s`

Corresponding constant:`mosek.dparam.basis_tol_s`**Description:**

Maximum absolute dual bound violation in an optimal basic solution.

Possible Values:

Any number between 1.0e-9 and +inf.

Default value:`1.0e-6`

- `basis_tol_x`

Corresponding constant:`mosek.dparam.basis_tol_x`**Description:**

Maximum absolute primal bound violation allowed in an optimal basic solution.

Possible Values:

Any number between 1.0e-9 and +inf.

Default value:`1.0e-6`

- `callback_freq`

Corresponding constant:

`mosek.dparam.callback_freq`

Description:

Controls the time between calls to the progress call-back function. Hence, if the value of this parameter is for example 10, then the call-back is called approximately each 10 seconds. A negative value is equivalent to infinity.

In general frequent call-backs may hurt the performance.

Possible Values:

Any number between -inf and +inf.

Default value:

-1.0

- `check_convexity_rel_tol`

Corresponding constant:

`mosek.dparam.check_convexity_rel_tol`

Description:

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:

$$d_i \leq -|Q_{ii}| * \text{check_convexity_rel_tol}$$

Possible Values:

Any number between 0 and +inf.

Default value:

1e-10

- `data_tol_ajj`

Corresponding constant:

`mosek.dparam.data_tol_ajj`

Description:

Absolute zero tolerance for elements in A . If any value A_{ij} is smaller than this parameter in absolute terms MOSEK will treat the values as zero and generate a warning.

Possible Values:

Any number between 1.0e-16 and 1.0e-6.

Default value:

1.0e-12

- `data_tol_ajj_huge`

Corresponding constant:`mosek.dparam.data_tol_aj_huge`**Description:**

An element in A which is larger than this value in absolute size causes an error.

Possible Values:

Any number between 0.0 and $+\infty$.

Default value:`1.0e20`

- `data_tol_aj_large`

Corresponding constant:`mosek.dparam.data_tol_aj_large`**Description:**

An element in A which is larger than this value in absolute size causes a warning message to be printed.

Possible Values:

Any number between 0.0 and $+\infty$.

Default value:`1.0e10`

- `data_tol_bound_inf`

Corresponding constant:`mosek.dparam.data_tol_bound_inf`**Description:**

Any bound which in absolute value is greater than this parameter is considered infinite.

Possible Values:

Any number between 0.0 and $+\infty$.

Default value:`1.0e16`

- `data_tol_bound_wrn`

Corresponding constant:`mosek.dparam.data_tol_bound_wrn`**Description:**

If a bound value is larger than this value in absolute size, then a warning message is issued.

Possible Values:

Any number between 0.0 and $+\infty$.

Default value:`1.0e8`

- `data_tol_c_huge`

Corresponding constant:

`mosek.dparam.data_tol_c_huge`

Description:

An element in c which is larger than the value of this parameter in absolute terms is considered to be huge and generates an error.

Possible Values:

Any number between 0.0 and $+\text{inf}$.

Default value:

1.0e16

- `data_tol_cj_large`

Corresponding constant:

`mosek.dparam.data_tol_cj_large`

Description:

An element in c which is larger than this value in absolute terms causes a warning message to be printed.

Possible Values:

Any number between 0.0 and $+\text{inf}$.

Default value:

1.0e8

- `data_tol_qij`

Corresponding constant:

`mosek.dparam.data_tol_qij`

Description:

Absolute zero tolerance for elements in Q matrices.

Possible Values:

Any number between 0.0 and $+\text{inf}$.

Default value:

1.0e-16

- `data_tol_x`

Corresponding constant:

`mosek.dparam.data_tol_x`

Description:

Zero tolerance for constraints and variables i.e. if the distance between the lower and upper bound is less than this value, then the lower and lower bound is considered identical.

Possible Values:

Any number between 0.0 and $+\text{inf}$.

Default value:

1.0e-8

- `feasrepair_tol`

Corresponding constant:`mosek.dparam.feasrepair_tol`**Description:**

Tolerance for constraint enforcing upper bound on sum of weighted violations in feasibility repair.

Possible Values:

Any number between $1.0\text{e-}16$ and $1.0\text{e+}16$.

Default value:`1.0e-10`

- `intpnt_co_tol_dfeas`

Corresponding constant:`mosek.dparam.intpnt_co_tol_dfeas`**Description:**

Dual feasibility tolerance used by the conic interior-point optimizer.

Possible Values:

Any number between 0.0 and 1.0.

Default value:`1.0e-8`**See also:**

`mosek.dparam.intpnt_co_tol_near_rel` Optimality tolerance for the conic solver.

- `intpnt_co_tol_infeas`

Corresponding constant:`mosek.dparam.intpnt_co_tol_infeas`**Description:**

Controls when the conic interior-point optimizer declares the model primal or dual infeasible. A small number means the optimizer gets more conservative about declaring the model infeasible.

Possible Values:

Any number between 0.0 and 1.0.

Default value:`1.0e-8`

- `intpnt_co_tol_mu_red`

Corresponding constant:`mosek.dparam.intpnt_co_tol_mu_red`**Description:**

Relative complementarity gap tolerance feasibility tolerance used by the conic interior-point optimizer.

Possible Values:

Any number between 0.0 and 1.0.

Default value:

1.0e-8

- `intpnt.co.tol.near_rel`

Corresponding constant:

`mosek.dparam.intpnt.co.tol.near_rel`

Description:

If MOSEK cannot compute a solution that has the prescribed accuracy, then it will multiply the termination tolerances with value of this parameter. If the solution then satisfies the termination criteria, then the solution is denoted near optimal, near feasible and so forth.

Possible Values:

Any number between 1.0 and +inf.

Default value:

100

- `intpnt.co.tol.pfeas`

Corresponding constant:

`mosek.dparam.intpnt.co.tol.pfeas`

Description:

Primal feasibility tolerance used by the conic interior-point optimizer.

Possible Values:

Any number between 0.0 and 1.0.

Default value:

1.0e-8

See also:

`mosek.dparam.intpnt.co.tol.near_rel` Optimality tolerance for the conic solver.

- `intpnt.co.tol.rel_gap`

Corresponding constant:

`mosek.dparam.intpnt.co.tol.rel_gap`

Description:

Relative gap termination tolerance used by the conic interior-point optimizer.

Possible Values:

Any number between 0.0 and 1.0.

Default value:

1.0e-8

See also:

`mosek.dparam.intpnt.co.tol.near_rel` Optimality tolerance for the conic solver.

- `intpnt_nl_merit_bal`

Corresponding constant:`mosek.dparam.intpnt_nl_merit_bal`**Description:**

Controls if the complementarity and infeasibility is converging to zero at about equal rates.

Possible Values:

Any number between 0.0 and 0.99.

Default value:`1.0e-4`

- `intpnt_nl_tol_dfeas`

Corresponding constant:`mosek.dparam.intpnt_nl_tol_dfeas`**Description:**

Dual feasibility tolerance used when a nonlinear model is solved.

Possible Values:

Any number between 0.0 and 1.0.

Default value:`1.0e-8`

- `intpnt_nl_tol_mu_red`

Corresponding constant:`mosek.dparam.intpnt_nl_tol_mu_red`**Description:**

Relative complementarity gap tolerance.

Possible Values:

Any number between 0.0 and 1.0.

Default value:`1.0e-12`

- `intpnt_nl_tol_near_rel`

Corresponding constant:`mosek.dparam.intpnt_nl_tol_near_rel`**Description:**

If the MOSEK nonlinear interior-point optimizer cannot compute a solution that has the prescribed accuracy, then it will multiply the termination tolerances with value of this parameter. If the solution then satisfies the termination criteria, then the solution is denoted near optimal, near feasible and so forth.

Possible Values:

Any number between 1.0 and $+\infty$.

Default value:

1000.0

• **intpnt_nl_tol_pfeas****Corresponding constant:**

mosek.dparam.intpnt_nl_tol_pfeas

Description:

Primal feasibility tolerance used when a nonlinear model is solved.

Possible Values:

Any number between 0.0 and 1.0.

Default value:

1.0e-8

• **intpnt_nl_tol_rel_gap****Corresponding constant:**

mosek.dparam.intpnt_nl_tol_rel_gap

Description:

Relative gap termination tolerance for nonlinear problems.

Possible Values:

Any number between 1.0e-14 and +inf.

Default value:

1.0e-6

• **intpnt_nl_tol_rel_step****Corresponding constant:**

mosek.dparam.intpnt_nl_tol_rel_step

Description:

Relative step size to the boundary for general nonlinear optimization problems.

Possible Values:

Any number between 1.0e-4 and 0.9999999.

Default value:

0.995

• **intpnt_tol_dfeas****Corresponding constant:**

mosek.dparam.intpnt_tol_dfeas

Description:

Dual feasibility tolerance used for linear and quadratic optimization problems.

Possible Values:

Any number between 0.0 and 1.0.

Default value:

1.0e-8

- `intpnt_tol_dsafe`

Corresponding constant:`mosek.dparam.intpnt_tol_dsafe`**Description:**

Controls the initial dual starting point used by the interior-point optimizer. If the interior-point optimizer converges slowly.

Possible Values:

Any number between 1.0e-4 and +inf.

Default value:

1.0

- `intpnt_tol_infeas`

Corresponding constant:`mosek.dparam.intpnt_tol_infeas`**Description:**

Controls when the optimizer declares the model primal or dual infeasible. A small number means the optimizer gets more conservative about declaring the model infeasible.

Possible Values:

Any number between 0.0 and 1.0.

Default value:

1.0e-8

- `intpnt_tol_mu_red`

Corresponding constant:`mosek.dparam.intpnt_tol_mu_red`**Description:**

Relative complementarity gap tolerance.

Possible Values:

Any number between 0.0 and 1.0.

Default value:

1.0e-16

- `intpnt_tol_path`

Corresponding constant:`mosek.dparam.intpnt_tol_path`**Description:**

Controls how close the interior-point optimizer follows the central path. A large value of this parameter means the central is followed very closely. On numerical unstable problems it may be worthwhile to increase this parameter.

Possible Values:

Any number between 0.0 and 0.9999.

Default value:

1.0e-8

- `intpnt_tol_pfeas`

Corresponding constant:

`mosek.dparam.intpnt_tol_pfeas`

Description:

Primal feasibility tolerance used for linear and quadratic optimization problems.

Possible Values:

Any number between 0.0 and 1.0.

Default value:

1.0e-8

- `intpnt_tol_psafe`

Corresponding constant:

`mosek.dparam.intpnt_tol_psafe`

Description:

Controls the initial primal starting point used by the interior-point optimizer. If the interior-point optimizer converges slowly and/or the constraint or variable bounds are very large, then it may be worthwhile to increase this value.

Possible Values:

Any number between 1.0e-4 and +inf.

Default value:

1.0

- `intpnt_tol_rel_gap`

Corresponding constant:

`mosek.dparam.intpnt_tol_rel_gap`

Description:

Relative gap termination tolerance.

Possible Values:

Any number between 1.0e-14 and +inf.

Default value:

1.0e-8

- `intpnt_tol_rel_step`

Corresponding constant:

`mosek.dparam.intpnt_tol_rel_step`

Description:

Relative step size to the boundary for linear and quadratic optimization problems.

Possible Values:

Any number between 1.0e-4 and 0.999999.

Default value:

0.9999

- `intpnt_tol_step_size`

Corresponding constant:`mosek.dparam.intpnt_tol_step_size`**Description:**

If the step size falls below the value of this parameter, then the interior-point optimizer assumes that it is stalled. If it does not make any progress.

Possible Values:

Any number between 0.0 and 1.0.

Default value:

1.0e-10

- `lower_obj_cut`

Corresponding constant:`mosek.dparam.lower_obj_cut`**Description:**

If either a primal or dual feasible solution is found proving that the optimal objective value is outside, the interval [`mosek.dparam.lower_obj_cut`, `mosek.dparam.upper_obj_cut`], then MOSEK is terminated.

Possible Values:

Any number between -inf and +inf.

Default value:

-1.0e30

See also:

`mosek.dparam.lower_obj_cut_finite_trh` Objective bound.

- `lower_obj_cut_finite_trh`

Corresponding constant:`mosek.dparam.lower_obj_cut_finite_trh`**Description:**

If the lower objective cut is less than the value of this parameter value, then the lower objective cut i.e. `mosek.dparam.lower_obj_cut` is treated as $-\infty$.

Possible Values:

Any number between -inf and +inf.

Default value:

-0.5e30

- `mio_disable_term_time`

Corresponding constant:`mosek.dparam.mio_disable_term_time`

Description:

The termination criteria governed by

- `mosek.iparam.mio_max_num_relaxs`
- `mosek.iparam.mio_max_num_branches`
- `mosek.dparam.mio_near_tol_abs_gap`
- `mosek.dparam.mio_near_tol_rel_gap`

is disabled the first n seconds. This parameter specifies the number n . A negative value is identical to infinity i.e. the termination criteria are never checked.

Possible Values:

Any number between $-\infty$ and $+\infty$.

Default value:

`-1.0`

See also:

- `mosek.iparam.mio_max_num_relaxs` Maximum number of relaxations in branch and bound search.
- `mosek.iparam.mio_max_num_branches` Maximum number of branches allowed during the branch and bound search.
- `mosek.dparam.mio_near_tol_abs_gap` Relaxed absolute optimality tolerance employed by the mixed-integer optimizer.
- `mosek.dparam.mio_near_tol_rel_gap` The mixed-integer optimizer is terminated when this tolerance is satisfied.

- `mio_heuristic_time`

Corresponding constant:

`mosek.dparam.mio_heuristic_time`

Description:

Minimum amount of time to be used in the heuristic search for a good feasible integer solution. A negative values implies that the optimizer decides the amount of time to be spent in the heuristic.

Possible Values:

Any number between $-\infty$ and $+\infty$.

Default value:

`-1.0`

- `mio_max_time`

Corresponding constant:

`mosek.dparam.mio_max_time`

Description:

This parameter limits the maximum time spent by the mixed-integer optimizer. A negative number means infinity.

Possible Values:

Any number between $-\infty$ and $+\infty$.

Default value:

-1.0

- `mio_max_time_aprx_opt`

Corresponding constant:

`mosek.dparam.mio_max_time_aprx_opt`

Description:

Number of seconds spent by the mixed-integer optimizer before the `mosek.dparam.mio_tol_rel_relax_int` is applied.

Possible Values:

Any number between 0.0 and +inf.

Default value:

60

- `mio_near_tol_abs_gap`

Corresponding constant:

`mosek.dparam.mio_near_tol_abs_gap`

Description:

Relaxed absolute optimality tolerance employed by the mixed-integer optimizer. This termination criteria is delayed. See `mosek.dparam.mio_disable_term_time` for details.

Possible Values:

Any number between 0.0 and +inf.

Default value:

0.0

See also:

`mosek.dparam.mio_disable_term_time` Certain termination criteria is disabled within the mixed-integer optimizer for period time specified by the parameter.

- `mio_near_tol_rel_gap`

Corresponding constant:

`mosek.dparam.mio_near_tol_rel_gap`

Description:

The mixed-integer optimizer is terminated when this tolerance is satisfied. This termination criteria is delayed. See `mosek.dparam.mio_disable_term_time` for details.

Possible Values:

Any number between 0.0 and +inf.

Default value:

1.0e-3

See also:

`mosek.dparam.mio_disable_term_time` Certain termination criteria is disabled within the mixed-integer optimizer for period time specified by the parameter.

- `mio_rel_add_cut_limited`

Corresponding constant:`mosek.dparam.mio_rel_add_cut_limited`**Description:**

Controls how many cuts the mixed-integer optimizer is allowed to add to the problem. Let α be the value of this parameter and m the number constraints, then mixed-integer optimizer is allowed to αm cuts.

Possible Values:

Any number between 0.0 and 2.0.

Default value:

0.75

- `mio_rel_gap_const`

Corresponding constant:`mosek.dparam.mio_rel_gap_const`**Description:**

This value is used to compute the relative gap for the solution to an integer optimization problem.

Possible Values:

Any number between 1.0e-15 and +inf.

Default value:

1.0e-10

- `mio_tol_abs_gap`

Corresponding constant:`mosek.dparam.mio_tol_abs_gap`**Description:**

Absolute optimality tolerance employed by the mixed-integer optimizer.

Possible Values:

Any number between 0.0 and +inf.

Default value:

0.0

- `mio_tol_abs_relax_int`

Corresponding constant:`mosek.dparam.mio_tol_abs_relax_int`**Description:**

Absolute relaxation tolerance of the integer constraints. I.e. $\min(|x| - \lfloor x \rfloor, \lceil x \rceil - |x|)$ is less than the tolerance then the integer restrictions assumed to be satisfied.

Possible Values:

Any number between 0.0 and +inf.

Default value:

1.0e-5

• `mio_tol_feas`**Corresponding constant:**`mosek.dparam.mio_tol_feas`**Description:**

Feasibility tolerance for mixed integer solver. Any solution with maximum infeasibility below this value will be considered feasible.

Possible Values:

Any number between 0.0 and +inf.

Default value:

1.0e-7

• `mio_tol_rel_gap`**Corresponding constant:**`mosek.dparam.mio_tol_rel_gap`**Description:**

Relative optimality tolerance employed by the mixed-integer optimizer.

Possible Values:

Any number between 0.0 and +inf.

Default value:

1.0e-4

• `mio_tol_rel_relax_int`**Corresponding constant:**`mosek.dparam.mio_tol_rel_relax_int`**Description:**

Relative relaxation tolerance of the integer constraints. I.e $(\min(|x| - \lfloor x \rfloor, \lceil x \rceil - |x|))$ is less than the tolerance times $|x|$ then the integer restrictions assumed to be satisfied.

Possible Values:

Any number between 0.0 and +inf.

Default value:

1.0e-6

• `mio_tol_x`**Corresponding constant:**`mosek.dparam.mio_tol_x`**Description:**

Absolute solution tolerance used in mixed-integer optimizer.

Possible Values:

Any number between 0.0 and +inf.

Default value:

1.0e-6

• **nonconvex_tol_feas****Corresponding constant:**`mosek.dparam.nonconvex_tol_feas`**Description:**

Feasibility tolerance used by the nonconvex optimizer.

Possible Values:

Any number between 0.0 and +inf.

Default value:

1.0e-6

• **nonconvex_tol_opt****Corresponding constant:**`mosek.dparam.nonconvex_tol_opt`**Description:**

Optimality tolerance used by the nonconvex optimizer.

Possible Values:

Any number between 0.0 and +inf.

Default value:

1.0e-7

• **optimizer_max_time****Corresponding constant:**`mosek.dparam.optimizer_max_time`**Description:**

Maximum amount of time the optimizer is allowed to spent on the optimization. A negative number means infinity.

Possible Values:

Any number between -inf and +inf.

Default value:

-1.0

• **presolve_tol_aij****Corresponding constant:**`mosek.dparam.presolve_tol_aij`**Description:**Absolute zero tolerance employed for a_{ij} in the presolve.**Possible Values:**

Any number between 1.0e-15 and +inf.

Default value:

1.0e-12

• **presolve_tol_lin_dep****Corresponding constant:**

mosek.dparam.presolve_tol_lin_dep

Description:

Controls when a constraint is determined to be linearly dependent.

Possible Values:

Any number between 0.0 and +inf.

Default value:

1.0e-6

• **presolve_tol_s****Corresponding constant:**

mosek.dparam.presolve_tol_s

Description:Absolute zero tolerance employed for s_i in the presolve.**Possible Values:**

Any number between 0.0 and +inf.

Default value:

1.0e-8

• **presolve_tol_x****Corresponding constant:**

mosek.dparam.presolve_tol_x

Description:Absolute zero tolerance employed for x_j in the presolve.**Possible Values:**

Any number between 0.0 and +inf.

Default value:

1.0e-8

• **qcqo_reformulate_rel_drop_tol****Corresponding constant:**

mosek.dparam.qcqo_reformulate_rel_drop_tol

Description:

This parameter determines when columns are dropped in incomplete cholesky factorization doing reformulation of quadratic problems.

Possible Values:

Any number between 0 and +inf.

Default value:

1e-15

- `sim_lu_tol_rel_piv`

Corresponding constant:`mosek.dparam.sim_lu_tol_rel_piv`**Description:**

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.

Possible Values:

Any number between 1.0e-6 and 0.999999.

Default value:

0.01

- `simplex_abs_tol_piv`

Corresponding constant:`mosek.dparam.simplex_abs_tol_piv`**Description:**

Absolute pivot tolerance employed by the simplex optimizers.

Possible Values:

Any number between 1.0e-12 and +inf.

Default value:

1.0e-7

- `upper_obj_cut`

Corresponding constant:`mosek.dparam.upper_obj_cut`**Description:**

If either a primal or dual feasible solution is found proving that the optimal objective value is outside, [`mosek.dparam.lower_obj_cut`, `mosek.dparam.upper_obj_cut`], then MOSEK is terminated.

Possible Values:

Any number between -inf and +inf.

Default value:

1.0e30

See also:

`mosek.dparam.upper_obj_cut_finite_trh` Objective bound.

- `upper_obj_cut_finite_trh`

Corresponding constant:

`mosek.dparam.upper_obj_cut_finite_trh`

Description:

If the upper objective cut is greater than the value of this value parameter, then the the upper objective cut `mosek.dparam.upper_obj_cut` is treated as ∞ .

Possible Values:

Any number between -inf and +inf.

Default value:

0.5e30

15.3 Integer parameters

- `mosek.iparam.alloc_add_qnz` 313
Controls how the quadratic matrixes are extended.
- `mosek.iparam.ana_sol_basis` 313
Controls whether the basis matrix is analyzed in solaution analyzer.
- `mosek.iparam.ana_sol_print_violated` 314
Controls whether a list of violated constraints is printed.
- `mosek.iparam.auto_sort_a_before_opt` 314
Controls whether the elements in each column of A are sorted before an optimization is performed.
- `mosek.iparam.auto_update_sol_info` 314
Controls whether the solution information items are automatically updated after an optimization is performed.
- `mosek.iparam.basis_solve_use_plus_one` 315
Controls the sign of the columns in the basis matrix corresponding to slack variables.
- `mosek.iparam.bi_clean_optimizer` 315
Controls which simplex optimizer is used in the clean-up phase.
- `mosek.iparam.bi_ignore_max_iter` 316
Turns on basis identification in case the interior-point optimizer is terminated due to maximum number of iterations.
- `mosek.iparam.bi_ignore_num_error` 316
Turns on basis identification in case the interior-point optimizer is terminated due to a numerical problem.
- `mosek.iparam.bi_max_iterations` 316
Maximum number of iterations after basis identification.
- `mosek.iparam.cache_license` 316
Control license caching.

- `mosek.iparam.cache_size_l1` 317
Specifies the size of the level 1 cache of the processor.
- `mosek.iparam.cache_size_l2` 317
Specifies the size of the level 2 cache of the processor.
- `mosek.iparam.check_convexity` 317
Specify the level of convexity check on quadratic problems
- `mosek.iparam.check_task_data` 318
If this feature is turned on, then the task data is checked for bad values i.e. NaNs. before an optimization is performed.
- `mosek.iparam.concurrent_num_optimizers` 318
The maximum number of simultaneous optimizations that will be started by the concurrent optimizer.
- `mosek.iparam.concurrent_priority_dual_simplex` 318
Priority of the dual simplex algorithm when selecting solvers for concurrent optimization.
- `mosek.iparam.concurrent_priority_free_simplex` 318
Priority of the free simplex optimizer when selecting solvers for concurrent optimization.
- `mosek.iparam.concurrent_priority_intpnt` 319
Priority of the interior-point algorithm when selecting solvers for concurrent optimization.
- `mosek.iparam.concurrent_priority_primal_simplex` 319
Priority of the primal simplex algorithm when selecting solvers for concurrent optimization.
- `mosek.iparam.cpu_type` 319
Specifies the CPU type.
- `mosek.iparam.data_check` 320
Enable data checking for debug purposes.
- `mosek.iparam.feasrepair_optimize` 320
Controls which type of feasibility analysis is to be performed.
- `mosek.iparam.infeas_generic_names` 320
Controls the contents of the infeasibility report.
- `mosek.iparam.infeas_prefer_primal` 321
Controls which certificate is used if both primal- and dual- certificate of infeasibility is available.
- `mosek.iparam.infeas_report_auto` 321
Turns the feasibility report on or off.
- `mosek.iparam.infeas_report_level` 321
Controls the contents of the infeasibility report.
- `mosek.iparam.intpnt_basis` 321
Controls whether basis identification is performed.

- `mosek.iparam.intpnt_diff_step` 322
Controls whether different step sizes are allowed in the primal and dual space.
- `mosek.iparam.intpnt_factor_debug_lvl` 322
Controls factorization debug level.
- `mosek.iparam.intpnt_factor_method` 323
Controls the method used to factor the Newton equation system.
- `mosek.iparam.intpnt_max_iterations` 323
Controls the maximum number of iterations allowed in the interior-point optimizer.
- `mosek.iparam.intpnt_max_num_cor` 323
Maximum number of correction steps.
- `mosek.iparam.intpnt_max_num_refinement_steps` 323
Maximum number of steps to be used by the iterative search direction refinement.
- `mosek.iparam.intpnt_num_threads` 324
Controls the number of threads employed by the interior-point optimizer. If set to a positive number MOSEK will use this number of threads. If zero the number of threads used will equal the number of cores detected on the machine.
- `mosek.iparam.intpnt_off_col_trh` 324
Controls the aggressiveness of the offending column detection.
- `mosek.iparam.intpnt_order_method` 324
Controls the ordering strategy.
- `mosek.iparam.intpnt_regularization_use` 325
Controls whether regularization is allowed.
- `mosek.iparam.intpnt_scaling` 325
Controls how the problem is scaled before the interior-point optimizer is used.
- `mosek.iparam.intpnt_solve_form` 325
Controls whether the primal or the dual problem is solved.
- `mosek.iparam.intpnt_starting_point` 326
Starting point used by the interior-point optimizer.
- `mosek.iparam.lic_trh_expiry_wrn` 326
Controls when expiry warnings are issued.
- `mosek.iparam.license_allow_overuse` 326
Controls if license overuse is allowed when caching licenses
- `mosek.iparam.license_cache_time` 326
Setting this parameter no longer has any effect.
- `mosek.iparam.license_check_time` 327
Controls the license manager client behavior.

- `mosek.iparam.license_debug` 327
Controls the license manager client debugging behavior.
- `mosek.iparam.license_pause_time` 327
Controls license manager client behavior.
- `mosek.iparam.license_suppress_expire_wrns` 328
Controls license manager client behavior.
- `mosek.iparam.license_wait` 328
Controls if MOSEK should queue for a license if none is available.
- `mosek.iparam.log` 328
Controls the amount of log information.
- `mosek.iparam.log_bi` 329
Controls the amount of output printed by the basis identification procedure. A higher level implies that more information is logged.
- `mosek.iparam.log_bi_freq` 329
Controls the logging frequency.
- `mosek.iparam.log_check_convexity` 329
Controls logging in convexity check on quadratic problems. Set to a positive value to turn logging on.

If a quadratic coefficient matrix is found to violate the requirement of PSD (NSD) then a list of negative (positive) pivot elements is printed. The absolute value of the pivot elements is also shown.
- `mosek.iparam.log_concurrent` 329
Controls amount of output printed by the concurrent optimizer.
- `mosek.iparam.log_cut_second_opt` 330
Controls the reduction in the log levels for the second and any subsequent optimizations.
- `mosek.iparam.log_factor` 330
If turned on, then the factor log lines are added to the log.
- `mosek.iparam.log_feasrepair` 330
Controls the amount of output printed when performing feasibility repair.
- `mosek.iparam.log_file` 331
If turned on, then some log info is printed when a file is written or read.
- `mosek.iparam.log_head` 331
If turned on, then a header line is added to the log.
- `mosek.iparam.log_infeas_ana` 331
Controls log level for the infeasibility analyzer.
- `mosek.iparam.log_intpnt` 331
Controls the amount of log information from the interior-point optimizers.

- `mosek.iparam.log_mio` 332
Controls the amount of log information from the mixed-integer optimizers.
- `mosek.iparam.log_mio_freq` 332
The mixed-integer solver logging frequency.
- `mosek.iparam.log_nonconvex` 332
Controls amount of output printed by the nonconvex optimizer.
- `mosek.iparam.log_optimizer` 332
Controls the amount of general optimizer information that is logged.
- `mosek.iparam.log_order` 333
If turned on, then factor lines are added to the log.
- `mosek.iparam.log_param` 333
Controls the amount of information printed out about parameter changes.
- `mosek.iparam.log_presolve` 333
Controls amount of output printed by the presolve procedure. A higher level implies that more information is logged.
- `mosek.iparam.log_response` 333
Controls amount of output printed when response codes are reported. A higher level implies that more information is logged.
- `mosek.iparam.log_sensitivity` 334
Control logging in sensitivity analyzer.
- `mosek.iparam.log_sensitivity_opt` 334
Control logging in sensitivity analyzer.
- `mosek.iparam.log_sim` 334
Controls the amount of log information from the simplex optimizers.
- `mosek.iparam.log_sim_freq` 335
Controls simplex logging frequency.
- `mosek.iparam.log_sim_minor` 335
Currently not in use.
- `mosek.iparam.log_sim_network_freq` 335
Controls the network simplex logging frequency.
- `mosek.iparam.log_storage` 335
Controls the memory related log information.
- `mosek.iparam.lp_write_ignore_incompatible_items` 336
Controls the result of writing a problem containing incompatible items to an LP file.
- `mosek.iparam.max_num_warnings` 336
Waning level. A higher value results in more warnings.

- `mosek.iparam.mio_branch_dir` 336
Controls whether the mixed-integer optimizer is branching up or down by default.
- `mosek.iparam.mio_branch_priorities_use` 336
Controls whether branching priorities are used by the mixed-integer optimizer.
- `mosek.iparam.mio_construct_sol` 337
Controls if an initial mixed integer solution should be constructed from the values of the integer variables.
- `mosek.iparam.mio_cont_sol` 337
Controls the meaning of interior-point and basic solutions in mixed integer problems.
- `mosek.iparam.mio_cut_level_root` 337
Controls the cut level employed by the mixed-integer optimizer at the root node.
- `mosek.iparam.mio_cut_level_tree` 338
Controls the cut level employed by the mixed-integer optimizer in the tree.
- `mosek.iparam.mio_feaspump_level` 338
Controls the feasibility pump heuristic which is used to construct a good initial feasible solution.
- `mosek.iparam.mio_heuristic_level` 339
Controls the heuristic employed by the mixed-integer optimizer to locate an initial integer feasible solution.
- `mosek.iparam.mio_hotstart` 339
Controls whether the integer optimizer is hot-started.
- `mosek.iparam.mio_keep_basis` 339
Controls whether the integer presolve keeps bases in memory.
- `mosek.iparam.mio_local_branch_number` 339
Controls the size of the local search space when doing local branching.
- `mosek.iparam.mio_max_num_branches` 340
Maximum number of branches allowed during the branch and bound search.
- `mosek.iparam.mio_max_num_relaxs` 340
Maximum number of relaxations in branch and bound search.
- `mosek.iparam.mio_max_num_solutions` 340
Controls how many feasible solutions the mixed-integer optimizer investigates.
- `mosek.iparam.mio_mode` 341
Turns on/off the mixed-integer mode.
- `mosek.iparam.mio_node_optimizer` 341
Controls which optimizer is employed at the non-root nodes in the mixed-integer optimizer.
- `mosek.iparam.mio_node_selection` 342
Controls the node selection strategy employed by the mixed-integer optimizer.

- `mosek.iparam.mio_optimizer_mode` 342
An experimental feature.
- `mosek.iparam.mio_presolve_aggregate` 342
Controls whether problem aggregation is performed in the mixed-integer presolve.
- `mosek.iparam.mio_presolve_probing` 343
Controls whether probing is employed by the mixed-integer presolve.
- `mosek.iparam.mio_presolve_use` 343
Controls whether presolve is performed by the mixed-integer optimizer.
- `mosek.iparam.mio_root_optimizer` 343
Controls which optimizer is employed at the root node in the mixed-integer optimizer.
- `mosek.iparam.mio_strong_branch` 344
The depth from the root in which strong branching is employed.
- `mosek.iparam.nonconvex_max_iterations` 344
Maximum number of iterations that can be used by the nonconvex optimizer.
- `mosek.iparam.objective_sense` 344
If the objective sense for the task is undefined, then the value of this parameter is used as the default objective sense.
- `mosek.iparam.opf_max_terms_per_line` 344
The maximum number of terms (linear and quadratic) per line when an OPF file is written.
- `mosek.iparam.opf_write_header` 345
Write a text header with date and MOSEK version in an OPF file.
- `mosek.iparam.opf_write_hints` 345
Write a hint section with problem dimensions in the beginning of an OPF file.
- `mosek.iparam.opf_write_parameters` 345
Write a parameter section in an OPF file.
- `mosek.iparam.opf_write_problem` 346
Write objective, constraints, bounds etc. to an OPF file.
- `mosek.iparam.opf_write_sol_bas` 346
Controls what is written to the OPF files.
- `mosek.iparam.opf_write_sol_itg` 346
Controls what is written to the OPF files.
- `mosek.iparam.opf_write_sol_itr` 346
Controls what is written to the OPF files.
- `mosek.iparam.opf_write_solutions` 347
Enable inclusion of solutions in the OPF files.

• <code>mosek.iparam.optimizer</code>	347
Controls which optimizer is used to optimize the task.	
• <code>mosek.iparam.param_read_case_name</code>	347
If turned on, then names in the parameter file are case sensitive.	
• <code>mosek.iparam.param_read_ign_error</code>	348
If turned on, then errors in parameter settings is ignored.	
• <code>mosek.iparam.presolve_elim_fill</code>	348
Maximum amount of fill-in in the elimination phase.	
• <code>mosek.iparam.presolve_eliminator_max_num_tries</code>	348
Control the maximum number of times the eliminator is tried.	
• <code>mosek.iparam.presolve_eliminator_use</code>	348
Controls whether free or implied free variables are eliminated from the problem.	
• <code>mosek.iparam.presolve_level</code>	349
Currently not used.	
• <code>mosek.iparam.presolve_lindep_use</code>	349
Controls whether the linear constraints are checked for linear dependencies.	
• <code>mosek.iparam.presolve_lindep_work_lim</code>	349
Controls linear dependency check in presolve.	
• <code>mosek.iparam.presolve_use</code>	350
Controls whether the presolve is applied to a problem before it is optimized.	
• <code>mosek.iparam.qo_separable_reformulation</code>	350
Determine if Quadratic programming problems should be reformulated to separable form.	
• <code>mosek.iparam.read_add_anz</code>	350
Controls how the constraint matrix is extended.	
• <code>mosek.iparam.read_add_con</code>	350
Additional number of constraints that is made room for in the problem.	
• <code>mosek.iparam.read_add_cone</code>	351
Additional number of conic constraints that is made room for in the problem.	
• <code>mosek.iparam.read_add_qnz</code>	351
Controls how the quadratic matrixes are extended.	
• <code>mosek.iparam.read_add_var</code>	351
Additional number of variables that is made room for in the problem.	
• <code>mosek.iparam.read_anz</code>	351
Controls the expected number of constraint non-zeros.	
• <code>mosek.iparam.read_con</code>	352
Controls the expected number of constraints.	

- `mosek.iparam.read_cone` 352
Controls the expected number of conic constraints.
- `mosek.iparam.read_data_compressed` 352
Controls the input file decompression.
- `mosek.iparam.read_data_format` 352
Format of the data file to be read.
- `mosek.iparam.read_keep_free_con` 353
Controls whether the free constraints are included in the problem.
- `mosek.iparam.read_lp_drop_new_vars_in_bou` 353
Controls how the LP files are interpreted.
- `mosek.iparam.read_lp_quoted_names` 353
If a name is in quotes when reading an LP file, the quotes will be removed.
- `mosek.iparam.read_mps_format` 354
Controls how strictly the MPS file reader interprets the MPS format.
- `mosek.iparam.read_mps_keep_int` 354
Controls if integer constraints are read.
- `mosek.iparam.read_mps_obj_sense` 354
Controls the MPS format extensions.
- `mosek.iparam.read_mps_quoted_names` 355
Controls the MPS format extensions.
- `mosek.iparam.read_mps_relax` 355
Controls the meaning of integer constraints.
- `mosek.iparam.read_mps_width` 355
Controls the maximal number of characters allowed in one line of the MPS file.
- `mosek.iparam.read_q_mode` 355
Controls how the Q matrices are read from the MPS file.
- `mosek.iparam.read_qnz` 356
Controls the expected number of quadratic non-zeros.
- `mosek.iparam.read_task_ignore_param` 356
Controls what information is used from the task files.
- `mosek.iparam.read_var` 356
Controls the expected number of variables.
- `mosek.iparam.sensitivity_all` 357
Controls sensitivity report behavior.
- `mosek.iparam.sensitivity_optimizer` 357
Controls which optimizer is used for optimal partition sensitivity analysis.

- `mosek.iparam.sensitivity_type` 358
Controls which type of sensitivity analysis is to be performed.
- `mosek.iparam.sim_basis_factor_use` 358
Controls whether a (LU) factorization of the basis is used in a hot-start. Forcing a refactorization sometimes improves the stability of the simplex optimizers, but in most cases there is a performance penalty.
- `mosek.iparam.sim_degen` 358
Controls how aggressively degeneration is handled.
- `mosek.iparam.sim_dual_crash` 359
Controls whether crashing is performed in the dual simplex optimizer.
- `mosek.iparam.sim_dual_phaseone_method` 359
An experimental feature.
- `mosek.iparam.sim_dual_restrict_selection` 359
Controls how aggressively restricted selection is used.
- `mosek.iparam.sim_dual_selection` 359
Controls the dual simplex strategy.
- `mosek.iparam.sim_exploit_dupvec` 360
Controls if the simplex optimizers are allowed to exploit duplicated columns.
- `mosek.iparam.sim_hotstart` 360
Controls the type of hot-start that the simplex optimizer perform.
- `mosek.iparam.sim_hotstart_lu` 361
Determines if the simplex optimizer should exploit the initial factorization.
- `mosek.iparam.sim_integer` 361
An experimental feature.
- `mosek.iparam.sim_max_iterations` 361
Maximum number of iterations that can be used by a simplex optimizer.
- `mosek.iparam.sim_max_num_setbacks` 361
Controls how many set-backs that are allowed within a simplex optimizer.
- `mosek.iparam.sim_network_detect` 362
Level of aggressiveness of network detection.
- `mosek.iparam.sim_network_detect_hotstart` 362
Level of aggressiveness of network detection in a simplex hot-start.
- `mosek.iparam.sim_network_detect_method` 362
Controls which type of detection method the network extraction should use.
- `mosek.iparam.sim_non_singular` 363
Controls if the simplex optimizer ensures a non-singular basis, if possible.

- `mosek.iparam.sim_primal_crash` 363
Controls the simplex crash.
- `mosek.iparam.sim_primal_phaseone_method` 363
An experimental feature.
- `mosek.iparam.sim_primal_restrict_selection` 363
Controls how aggressively restricted selection is used.
- `mosek.iparam.sim_primal_selection` 364
Controls the primal simplex strategy.
- `mosek.iparam.sim_refactor_freq` 364
Controls the basis refactoring frequency.
- `mosek.iparam.sim_reformulation` 365
Controls if the simplex optimizers are allowed to reformulate the problem.
- `mosek.iparam.sim_save_lu` 365
Controls if the LU factorization stored should be replaced with the LU factorization corresponding to the initial basis.
- `mosek.iparam.sim_scaling` 365
Controls how much effort is used in scaling the problem before a simplex optimizer is used.
- `mosek.iparam.sim_scaling_method` 366
Controls how the problem is scaled before a simplex optimizer is used.
- `mosek.iparam.sim_solve_form` 366
Controls whether the primal or the dual problem is solved by the primal-/dual- simplex optimizer.
- `mosek.iparam.sim_stability_priority` 366
Controls how high priority the numerical stability should be given.
- `mosek.iparam.sim_switch_optimizer` 366
Controls the simplex behavior.
- `mosek.iparam.sol_filter_keep_basic` 367
Controls the license manager client behavior.
- `mosek.iparam.sol_filter_keep_ranged` 367
Control the contents of the solution files.
- `mosek.iparam.sol_quoted_names` 367
Controls the solution file format.
- `mosek.iparam.sol_read_name_width` 368
Controls the input solution file format.
- `mosek.iparam.sol_read_width` 368
Controls the input solution file format.

- `mosek.iparam.solution_callback` 368
Indicates whether solution call-backs will be performed during the optimization.
- `mosek.iparam.timing_level` 369
Controls the amount of timing performed inside MOSEK.
- `mosek.iparam.warning_level` 369
Warning level.
- `mosek.iparam.write_bas_constraints` 369
Controls the basic solution file format.
- `mosek.iparam.write_bas_head` 369
Controls the basic solution file format.
- `mosek.iparam.write_bas_variables` 370
Controls the basic solution file format.
- `mosek.iparam.write_data_compressed` 370
Controls output file compression.
- `mosek.iparam.write_data_format` 370
Controls the output file format.
- `mosek.iparam.write_data_param` 371
Controls output file data.
- `mosek.iparam.write_free_con` 371
Controls the output file data.
- `mosek.iparam.write_generic_names` 371
Controls the output file data.
- `mosek.iparam.write_generic_names_io` 371
Index origin used in generic names.
- `mosek.iparam.write_int_constraints` 372
Controls the integer solution file format.
- `mosek.iparam.write_int_head` 372
Controls the integer solution file format.
- `mosek.iparam.write_int_variables` 372
Controls the integer solution file format.
- `mosek.iparam.write_lp_line_width` 373
Controls the LP output file format.
- `mosek.iparam.write_lp_quoted_names` 373
Controls LP output file format.
- `mosek.iparam.write_lp_strict_format` 373
Controls whether LP output files satisfy the LP format strictly.

- `mosek.iparam.write_lp_terms_per_line` 373
Controls the LP output file format.
- `mosek.iparam.write_mps_int` 374
Controls the output file data.
- `mosek.iparam.write_mps_obj_sense` 374
Controls the output file data.
- `mosek.iparam.write_mps_quoted_names` 374
Controls the output file data.
- `mosek.iparam.write_mps_strict` 374
Controls the output MPS file format.
- `mosek.iparam.write_precision` 375
Controls data precision employed in when writing an MPS file.
- `mosek.iparam.write_sol_constraints` 375
Controls the solution file format.
- `mosek.iparam.write_sol_head` 375
Controls solution file format.
- `mosek.iparam.write_sol_variables` 376
Controls the solution file format.
- `mosek.iparam.write_task_inc_sol` 376
Controls whether the solutions are stored in the task file too.
- `mosek.iparam.write_xml_mode` 376
Controls if linear coefficients should be written by row or column when writing in the XML file format.

- `alloc_add_qnz`

Corresponding constant:

`mosek.iparam.alloc_add_qnz`

Description:

Additional number of Q non-zeros that are allocated space for when `numanz` exceeds `maxnumqnz` during addition of new Q entries.

Possible Values:

Any number between 0 and `+inf`.

Default value:

5000

- `ana_sol_basis`

Corresponding constant:

`mosek.iparam.ana_sol_basis`

Description:

Controls whether the basis matrix is analyzed in solution analyzer.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.on`

- `ana_sol_print_violated`

Corresponding constant:

`mosek.iparam.ana_sol_print_violated`

Description:

Controls whether a list of violated constraints is printed when calling `Task.analyzesolution`.

All constraints violated by more than the value set by the parameter `mosek.dparam.ana_sol_infeas_tol` will be printed.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.off`

- `auto_sort_a_before_opt`

Corresponding constant:

`mosek.iparam.auto_sort_a_before_opt`

Description:

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.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.off`

- `auto_update_sol_info`

Corresponding constant:

`mosek.iparam.auto_update_sol_info`

Description:

Controls whether the solution information items are automatically updated after an optimization is performed.

Possible values:

`mosek.onoffkey.on` Switch the option on.
`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.on`

- `basis.solve.use_plus_one`

Corresponding constant:

`mosek.iparam.basis_solve_use_plus_one`

Description:

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 `mosek.onoffkey.on`, -1 is replaced by 1.

This has significance for the results returned by the `Task.solvewithbasis` function.

Possible values:

`mosek.onoffkey.on` Switch the option on.
`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.off`

- `bi_clean_optimizer`

Corresponding constant:

`mosek.iparam.bi_clean_optimizer`

Description:

Controls which simplex optimizer is used in the clean-up phase.

Possible values:

`mosek.optimizertype.intpnt` The interior-point optimizer is used.
`mosek.optimizertype.concurrent` The optimizer for nonconvex nonlinear problems.
`mosek.optimizertype.mixed_int` The mixed-integer optimizer.
`mosek.optimizertype.dual_simplex` The dual simplex optimizer is used.
`mosek.optimizertype.free` The optimizer is chosen automatically.
`mosek.optimizertype.primal_dual_simplex` The primal dual simplex optimizer is used.
`mosek.optimizertype.conic` The optimizer for problems having conic constraints.
`mosek.optimizertype.nonconvex` The optimizer for nonconvex nonlinear problems.
`mosek.optimizertype.qcone` For internal use only.
`mosek.optimizertype.primal_simplex` The primal simplex optimizer is used.
`mosek.optimizertype.free_simplex` One of the simplex optimizers is used.

Default value:

`mosek.optimizertype.free`

- `bi_ignore_max_iter`

Corresponding constant:

`mosek.iparam.bi_ignore_max_iter`

Description:

If the parameter `mosek.iparam.intpnt_basis` has the value `mosek.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 `mosek.onoffkey.on`.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.off`

- `bi_ignore_num_error`

Corresponding constant:

`mosek.iparam.bi_ignore_num_error`

Description:

If the parameter `mosek.iparam.intpnt_basis` has the value `mosek.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 `mosek.onoffkey.on`.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.off`

- `bi_max_iterations`

Corresponding constant:

`mosek.iparam.bi_max_iterations`

Description:

Controls the maximum number of simplex iterations allowed to optimize a basis after the basis identification.

Possible Values:

Any number between 0 and $+\infty$.

Default value:

1000000

- `cache_license`

Corresponding constant:

`mosek.iparam.cache_license`

Description:

Specifies if the license is kept checked out for the lifetime of the mosek environment (on) or returned to the server immediately after the optimization (off).

Check-in and check-out of licenses have an overhead. Frequent communication with the license server should be avoided.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.on`

- `cache_size_l1`

Corresponding constant:

`mosek.iparam.cache_size_l1`

Description:

Specifies the size of the cache of the computer. This parameter is potentially very important for the efficiency on computers if MOSEK cannot determine the cache size automatically.

If the cache size is negative, then MOSEK tries to determine the value automatically.

Possible Values:

Any number between $-\infty$ and $+\infty$.

Default value:

-1

- `cache_size_l2`

Corresponding constant:

`mosek.iparam.cache_size_l2`

Description:

Specifies the size of the cache of the computer. This parameter is potentially very important for the efficiency on computers where MOSEK cannot determine the cache size automatically.

If the cache size is negative, then MOSEK tries to determine the value automatically.

Possible Values:

Any number between $-\infty$ and $+\infty$.

Default value:

-1

- `check_convexity`

Corresponding constant:

`mosek.iparam.check_convexity`

Description:

Specify the level of convexity check on quadratic problems

Possible values:

`mosek.checkconvexitytype.simple` Perform simple and fast convexity check.

`mosek.checkconvexitytype.none` No convexity check.

`mosek.checkconvexitytype.full` Perform a full convexity check.

Default value:

`mosek.checkconvexitytype.full`

- `check_task_data`

Corresponding constant:

`mosek.iparam.check_task_data`

Description:

If this feature is turned on, then the task data is checked for bad values i.e. NaNs. before an optimization is performed.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.on`

- `concurrent_num_optimizers`

Corresponding constant:

`mosek.iparam.concurrent_num_optimizers`

Description:

The maximum number of simultaneous optimizations that will be started by the concurrent optimizer.

Possible Values:

Any number between 0 and +inf.

Default value:

2

- `concurrent_priority_dual_simplex`

Corresponding constant:

`mosek.iparam.concurrent_priority_dual_simplex`

Description:

Priority of the dual simplex algorithm when selecting solvers for concurrent optimization.

Possible Values:

Any number between 0 and +inf.

Default value:

2

- `concurrent_priority_free_simplex`

Corresponding constant:

`mosek.iparam.concurrent_priority_free_simplex`

Description:

Priority of the free simplex optimizer when selecting solvers for concurrent optimization.

Possible Values:

Any number between 0 and +inf.

Default value:

3

- `concurrent_priority_intpnt`

Corresponding constant:

`mosek.iparam.concurrent_priority_intpnt`

Description:

Priority of the interior-point algorithm when selecting solvers for concurrent optimization.

Possible Values:

Any number between 0 and +inf.

Default value:

4

- `concurrent_priority_primal_simplex`

Corresponding constant:

`mosek.iparam.concurrent_priority_primal_simplex`

Description:

Priority of the primal simplex algorithm when selecting solvers for concurrent optimization.

Possible Values:

Any number between 0 and +inf.

Default value:

1

- `cpu_type`

Corresponding constant:

`mosek.iparam.cpu_type`

Description:

Specifies the CPU type. By default MOSEK tries to auto detect the CPU type. Therefore, we recommend to change this parameter only if the auto detection does not work properly.

Possible values:

`mosek.cputype.powerpc_g5` A G5 PowerPC CPU.

`mosek.cputype.intel_pm` An Intel PM cpu.

`mosek.cputype.generic` An generic CPU type for the platform

`mosek.cputype.unknown` An unknown CPU.

`mosek.cputype.amd.opteron` An AMD Opteron (64 bit).

`mosek.cputype.intel_itanium2` An Intel Itanium2.
`mosek.cputype.amd_athlon` An AMD Athlon.
`mosek.cputype.hp_parisc20` An HP PA RISC version 2.0 CPU.
`mosek.cputype.intel_p4` An Intel Pentium P4 or Intel Xeon.
`mosek.cputype.intel_p3` An Intel Pentium P3.
`mosek.cputype.intel_core2` An Intel CORE2 cpu.

Default value:

`mosek.cputype.unknown`

- `data_check`

Corresponding constant:

`mosek.iparam.data_check`

Description:

If this option is turned on, then extensive data checking is enabled. It will slow down MOSEK but on the other hand help locating bugs.

Possible values:

`mosek.onoffkey.on` Switch the option on.
`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.on`

- `feasrepair_optimize`

Corresponding constant:

`mosek.iparam.feasrepair_optimize`

Description:

Controls which type of feasibility analysis is to be performed.

Possible values:

`mosek.feasrepairtype.optimize_none` Do not optimize the feasibility repair problem.
`mosek.feasrepairtype.optimize_combined` Minimize with original objective subject to minimal weighted violation of bounds.
`mosek.feasrepairtype.optimize_penalty` Minimize weighted sum of violations.

Default value:

`mosek.feasrepairtype.optimize_none`

- `infeas_generic_names`

Corresponding constant:

`mosek.iparam.infeas_generic_names`

Description:

Controls whether generic names are used when an infeasible subproblem is created.

Possible values:

`mosek.onoffkey.on` Switch the option on.
`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.off`

- `infeas_prefer_primal`

Corresponding constant:

`mosek.iparam.infeas_prefer_primal`

Description:

If both certificates of primal and dual infeasibility are supplied then only the primal is used when this option is turned on.

Possible values:

`mosek.onoffkey.on` Switch the option on.
`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.on`

- `infeas_report_auto`

Corresponding constant:

`mosek.iparam.infeas_report_auto`

Description:

Controls whether an infeasibility report is automatically produced after the optimization if the problem is primal or dual infeasible.

Possible values:

`mosek.onoffkey.on` Switch the option on.
`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.off`

- `infeas_report_level`

Corresponding constant:

`mosek.iparam.infeas_report_level`

Description:

Controls the amount of information presented in an infeasibility report. Higher values imply more information.

Possible Values:

Any number between 0 and $+\infty$.

Default value:

1

- `intpnt_basis`

Corresponding constant:

`mosek.iparam.intpnt_basis`

Description:

Controls whether the interior-point optimizer also computes an optimal basis.

Possible values:

`mosek.basindtype.always` Basis identification is always performed even if the interior-point optimizer terminates abnormally.

`mosek.basindtype.no_error` Basis identification is performed if the interior-point optimizer terminates without an error.

`mosek.basindtype.never` Never do basis identification.

`mosek.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.

`mosek.basindtype.other` Try another BI method.

Default value:

`mosek.basindtype.always`

See also:

`mosek.iparam.bi_ignore_max_iter` Turns on basis identification in case the interior-point optimizer is terminated due to maximum number of iterations.

`mosek.iparam.bi_ignore_num_error` Turns on basis identification in case the interior-point optimizer is terminated due to a numerical problem.

- `intpnt_diff_step`

Corresponding constant:

`mosek.iparam.intpnt_diff_step`

Description:

Controls whether different step sizes are allowed in the primal and dual space.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.on`

- `intpnt_factor_debug_lvl`

Corresponding constant:

`mosek.iparam.intpnt_factor_debug_lvl`

Description:

Controls factorization debug level.

Possible Values:

Any number between 0 and +inf.

Default value:

0

- `intpnt_factor_method`

Corresponding constant:

`mosek.iparam.intpnt_factor_method`

Description:

Controls the method used to factor the Newton equation system.

Possible Values:

Any number between 0 and +inf.

Default value:

0

- `intpnt_max_iterations`

Corresponding constant:

`mosek.iparam.intpnt_max_iterations`

Description:

Controls the maximum number of iterations allowed in the interior-point optimizer.

Possible Values:

Any number between 0 and +inf.

Default value:

400

- `intpnt_max_num_cor`

Corresponding constant:

`mosek.iparam.intpnt_max_num_cor`

Description:

Controls the maximum number of correctors allowed by the multiple corrector procedure.
A negative value means that MOSEK is making the choice.

Possible Values:

Any number between -1 and +inf.

Default value:

-1

- `intpnt_max_num_refinement_steps`

Corresponding constant:

`mosek.iparam.intpnt_max_num_refinement_steps`

Description:

Maximum number of steps to be used by the iterative refinement of the search direction.
A negative value implies that the optimizer Chooses the maximum number of iterative refinement steps.

Possible Values:

Any number between -inf and +inf.

Default value:

-1

- `intpnt_num_threads`

Corresponding constant:

`mosek.iparam.intpnt_num_threads`

Description:

Controls the number of threads employed by the interior-point optimizer. If set to a positive number MOSEK will use this number of threads. If zero the number of threads used will equal the number of cores detected on the machine.

Possible Values:

Any integer greater or equal to 0.

Default value:

1

- `intpnt_off_col_trh`

Corresponding constant:

`mosek.iparam.intpnt_off_col_trh`

Description:

Controls how many offending columns are detected in the Jacobian of the constraint matrix. 1 means aggressive detection, higher values mean less aggressive detection. 0 means no detection.

Possible Values:

Any number between 0 and +inf.

Default value:

40

- `intpnt_order_method`

Corresponding constant:

`mosek.iparam.intpnt_order_method`

Description:

Controls the ordering strategy used by the interior-point optimizer when factorizing the Newton equation system.

Possible values:

`mosek.orderingtype.none` No ordering is used.

`mosek.orderingtype.appminloc2` A variant of the approximate minimum local-fill-in ordering is used.

`mosek.orderingtype.appminloc1` Approximate minimum local-fill-in ordering is used.

`mosek.orderingtype.graphpar2` An alternative graph partitioning based ordering.

`mosek.orderingtype.free` The ordering method is chosen automatically.
`mosek.orderingtype.graphpar1` Graph partitioning based ordering.

Default value:

`mosek.orderingtype.free`

- `intpnt_regularization_use`

Corresponding constant:

`mosek.iparam.intpnt_regularization_use`

Description:

Controls whether regularization is allowed.

Possible values:

`mosek.onoffkey.on` Switch the option on.
`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.on`

- `intpnt_scaling`

Corresponding constant:

`mosek.iparam.intpnt_scaling`

Description:

Controls how the problem is scaled before the interior-point optimizer is used.

Possible values:

`mosek.scalingtype.none` No scaling is performed.
`mosek.scalingtype.moderate` A conservative scaling is performed.
`mosek.scalingtype.aggressive` A very aggressive scaling is performed.
`mosek.scalingtype.free` The optimizer chooses the scaling heuristic.

Default value:

`mosek.scalingtype.free`

- `intpnt_solve_form`

Corresponding constant:

`mosek.iparam.intpnt_solve_form`

Description:

Controls whether the primal or the dual problem is solved.

Possible values:

`mosek.solveform.primal` The optimizer should solve the primal problem.
`mosek.solveform.dual` The optimizer should solve the dual problem.
`mosek.solveform.free` The optimizer is free to solve either the primal or the dual problem.

Default value:

`mosek.solveform.free`

- `intpnt_starting_point`

Corresponding constant:

`mosek.iparam.intpnt_starting_point`

Description:

Starting point used by the interior-point optimizer.

Possible values:

`mosek.startpointtype.guess` The optimizer guesses a starting point.

`mosek.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.

`mosek.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.

`mosek.startpointtype.free` The starting point is chosen automatically.

Default value:

`mosek.startpointtype.free`

- `lic_trh_expiry_wrn`

Corresponding constant:

`mosek.iparam.lic_trh_expiry_wrn`

Description:

If a license feature expires in a number of days less than the value of this parameter then a warning will be issued.

Possible Values:

Any number between 0 and $+\infty$.

Default value:

7

- `license_allow_overuse`

Corresponding constant:

`mosek.iparam.license_allow_overuse`

Description:

Controls if license overuse is allowed when caching licenses

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.on`

- `license_cache_time`

Corresponding constant:`mosek.iparam.license_cache_time`**Description:**

Setting this parameter no longer has any effect. Please see `mosek.iparam.cache_license` for an alternative.

Possible Values:

Any number between 0 and 65555.

Default value:

5

- `license_check_time`

Corresponding constant:`mosek.iparam.license_check_time`**Description:**

The parameter specifies the number of seconds between the checks of all the active licenses in the MOSEK environment license cache. These checks are performed to determine if the licenses should be returned to the server.

Possible Values:

Any number between 1 and 120.

Default value:

1

- `license_debug`

Corresponding constant:`mosek.iparam.license_debug`**Description:**

This option is used to turn on debugging of the incense manager.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.off`

- `license_pause_time`

Corresponding constant:`mosek.iparam.license_pause_time`**Description:**

If `mosek.iparam.license_wait=mosek.onoffkey.on` and no license is available, then MOSEK sleeps a number of milliseconds between each check of whether a license has become free.

Possible Values:

Any number between 0 and 1000000.

Default value:

100

• **license_suppress_expire_wrns****Corresponding constant:**`mosek.iparam.license_suppress_expire_wrns`**Description:**

Controls whether license features expire warnings are suppressed.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:`mosek.onoffkey.off`• **license_wait****Corresponding constant:**`mosek.iparam.license_wait`**Description:**

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.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:`mosek.onoffkey.off`• **log****Corresponding constant:**`mosek.iparam.log`**Description:**

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 `mosek.iparam.log_cut_second_opt` for the second and any subsequent optimizations.

Possible Values:

Any number between 0 and +inf.

Default value:

10

See also:

`mosek.iparam.log_cut_second_opt` Controls the reduction in the log levels for the second and any subsequent optimizations.

- `log_bi`

Corresponding constant:

`mosek.iparam.log_bi`

Description:

Controls the amount of output printed by the basis identification procedure. A higher level implies that more information is logged.

Possible Values:

Any number between 0 and $+\infty$.

Default value:

4

- `log_bi_freq`

Corresponding constant:

`mosek.iparam.log_bi_freq`

Description:

Controls how frequent the optimizer outputs information about the basis identification and how frequent the user-defined call-back function is called.

Possible Values:

Any number between 0 and $+\infty$.

Default value:

2500

- `log_check_convexity`

Corresponding constant:

`mosek.iparam.log_check_convexity`

Description:

Controls logging in convexity check on quadratic problems. Set to a positive value to turn logging on.

If a quadratic coefficient matrix is found to violate the requirement of PSD (NSD) then a list of negative (positive) pivot elements is printed. The absolute value of the pivot elements is also shown.

Possible Values:

Any number between 0 and $+\infty$.

Default value:

0

- `log_concurrent`

Corresponding constant:

`mosek.iparam.log_concurrent`

Description:

Controls amount of output printed by the concurrent optimizer.

Possible Values:

Any number between 0 and +inf.

Default value:

1

- `log_cut_second_opt`

Corresponding constant:

`mosek.iparam.log_cut_second_opt`

Description:

If a task is employed to solve a sequence of optimization problems, then the value of the log levels is reduced by the value of this parameter. E.g `mosek.iparam.log` and `mosek.iparam.log_sim` are reduced by the value of this parameter for the second and any subsequent optimizations.

Possible Values:

Any number between 0 and +inf.

Default value:

1

See also:

`mosek.iparam.log` Controls the amount of log information.

`mosek.iparam.log_intpnt` Controls the amount of log information from the interior-point optimizers.

`mosek.iparam.log_mio` Controls the amount of log information from the mixed-integer optimizers.

`mosek.iparam.log_sim` Controls the amount of log information from the simplex optimizers.

- `log_factor`

Corresponding constant:

`mosek.iparam.log_factor`

Description:

If turned on, then the factor log lines are added to the log.

Possible Values:

Any number between 0 and +inf.

Default value:

1

- `log_feasrepair`

Corresponding constant:

`mosek.iparam.log_feasrepair`

Description:

Controls the amount of output printed when performing feasibility repair.

Possible Values:

Any number between 0 and +inf.

Default value:

0

- `log_file`

Corresponding constant:

`mosek.iparam.log_file`

Description:

If turned on, then some log info is printed when a file is written or read.

Possible Values:

Any number between 0 and +inf.

Default value:

1

- `log_head`

Corresponding constant:

`mosek.iparam.log_head`

Description:

If turned on, then a header line is added to the log.

Possible Values:

Any number between 0 and +inf.

Default value:

1

- `log_infeas_ana`

Corresponding constant:

`mosek.iparam.log_infeas_ana`

Description:

Controls amount of output printed by the infeasibility analyzer procedures. A higher level implies that more information is logged.

Possible Values:

Any number between 0 and +inf.

Default value:

1

- `log_intpnt`

Corresponding constant:

`mosek.iparam.log_intpnt`

Description:

Controls amount of output printed by the interior-point optimizer. A higher level implies that more information is logged.

Possible Values:

Any number between 0 and $+\infty$.

Default value:

4

- `log_mio`

Corresponding constant:

`mosek.iparam.log_mio`

Description:

Controls the log level for the mixed-integer optimizer. A higher level implies that more information is logged.

Possible Values:

Any number between 0 and $+\infty$.

Default value:

4

- `log_mio_freq`

Corresponding constant:

`mosek.iparam.log_mio_freq`

Description:

Controls how frequent the mixed-integer optimizer prints the log line. It will print line every time `mosek.iparam.log_mio_freq` relaxations have been solved.

Possible Values:

A integer value.

Default value:

1000

- `log_nonconvex`

Corresponding constant:

`mosek.iparam.log_nonconvex`

Description:

Controls amount of output printed by the nonconvex optimizer.

Possible Values:

Any number between 0 and $+\infty$.

Default value:

1

- `log_optimizer`

Corresponding constant:`mosek.iparam.log_optimizer`**Description:**

Controls the amount of general optimizer information that is logged.

Possible Values:

Any number between 0 and +inf.

Default value:

1

- `log_order`

Corresponding constant:`mosek.iparam.log_order`**Description:**

If turned on, then factor lines are added to the log.

Possible Values:

Any number between 0 and +inf.

Default value:

1

- `log_param`

Corresponding constant:`mosek.iparam.log_param`**Description:**

Controls the amount of information printed out about parameter changes.

Possible Values:

Any number between 0 and +inf.

Default value:

0

- `log_presolve`

Corresponding constant:`mosek.iparam.log_presolve`**Description:**

Controls amount of output printed by the presolve procedure. A higher level implies that more information is logged.

Possible Values:

Any number between 0 and +inf.

Default value:

1

- `log_response`

Corresponding constant:

`mosek.iparam.log_response`

Description:

Controls amount of output printed when response codes are reported. A higher level implies that more information is logged.

Possible Values:

Any number between 0 and +inf.

Default value:

0

- `log_sensitivity`

Corresponding constant:

`mosek.iparam.log_sensitivity`

Description:

Controls the amount of logging during the sensitivity analysis. 0: Means no logging information is produced. 1: Timing information is printed. 2: Sensitivity results are printed.

Possible Values:

Any number between 0 and +inf.

Default value:

1

- `log_sensitivity_opt`

Corresponding constant:

`mosek.iparam.log_sensitivity_opt`

Description:

Controls the amount of logging from the optimizers employed during the sensitivity analysis. 0 means no logging information is produced.

Possible Values:

Any number between 0 and +inf.

Default value:

0

- `log_sim`

Corresponding constant:

`mosek.iparam.log_sim`

Description:

Controls amount of output printed by the simplex optimizer. A higher level implies that more information is logged.

Possible Values:

Any number between 0 and +inf.

Default value:

4

- `log_sim_freq`

Corresponding constant:

`mosek.iparam.log_sim_freq`

Description:

Controls how frequent the simplex optimizer outputs information about the optimization and how frequent the user-defined call-back function is called.

Possible Values:

Any number between 0 and +inf.

Default value:

500

- `log_sim_minor`

Corresponding constant:

`mosek.iparam.log_sim_minor`

Description:

Currently not in use.

Possible Values:

Any number between 0 and +inf.

Default value:

1

- `log_sim_network_freq`

Corresponding constant:

`mosek.iparam.log_sim_network_freq`

Description:

Controls how frequent the network simplex optimizer outputs information about the optimization and how frequent the user-defined call-back function is called. The network optimizer will use a logging frequency equal to `mosek.iparam.log_sim_freq` times `mosek.iparam.log_sim_network`.

Possible Values:

Any number between 0 and +inf.

Default value:

50

- `log_storage`

Corresponding constant:

`mosek.iparam.log_storage`

Description:

When turned on, MOSEK prints messages regarding the storage usage and allocation.

Possible Values:

Any number between 0 and +inf.

Default value:

0

• `lp_write_ignore_incompatible_items`**Corresponding constant:**`mosek.iparam.lp_write_ignore_incompatible_items`**Description:**

Controls the result of writing a problem containing incompatible items to an LP file.

Possible values:`mosek.onoffkey.on` Switch the option on.`mosek.onoffkey.off` Switch the option off.**Default value:**`mosek.onoffkey.off`• `max_num_warnings`**Corresponding constant:**`mosek.iparam.max_num_warnings`**Description:**

Warning level. A higher value results in more warnings.

Possible Values:

Any number between 0 and +inf.

Default value:

10

• `mio_branch_dir`**Corresponding constant:**`mosek.iparam.mio_branch_dir`**Description:**

Controls whether the mixed-integer optimizer is branching up or down by default.

Possible values:`mosek.branchdir.down` The mixed-integer optimizer always chooses the down branch first.`mosek.branchdir.up` The mixed-integer optimizer always chooses the up branch first.`mosek.branchdir.free` The mixed-integer optimizer decides which branch to choose.**Default value:**`mosek.branchdir.free`• `mio_branch_priorities_use`**Corresponding constant:**`mosek.iparam.mio_branch_priorities_use`**Description:**

Controls whether branching priorities are used by the mixed-integer optimizer.

Possible values:

`mosek.onoffkey.on` Switch the option on.
`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.on`

- `mio_construct_sol`

Corresponding constant:

`mosek.iparam.mio_construct_sol`

Description:

If set to `mosek.onoffkey.on` and all integer variables have been given a value for which a feasible mixed integer solution exists, then MOSEK generates an initial solution to the mixed integer problem by fixing all integer values and solving the remaining problem.

Possible values:

`mosek.onoffkey.on` Switch the option on.
`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.off`

- `mio_cont_sol`

Corresponding constant:

`mosek.iparam.mio_cont_sol`

Description:

Controls the meaning of the interior-point and basic solutions in mixed integer problems.

Possible values:

`mosek.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.
`mosek.miocontsoltype.none` No interior-point or basic solution are reported when the mixed-integer optimizer is used.
`mosek.miocontsoltype.root` The reported interior-point and basic solutions are a solution to the root node problem when mixed-integer optimizer is used.
`mosek.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.

Default value:

`mosek.miocontsoltype.none`

- `mio_cut_level_root`

Corresponding constant:

`mosek.iparam.mio_cut_level_root`

Description:

Controls the cut level employed by the mixed-integer optimizer at the root node. A negative value means a default value determined by the mixed-integer optimizer is used. By adding the appropriate values from the following table the employed cut types can be controlled.

GUB cover	+2
Flow cover	+4
Lifting	+8
Plant location	+16
Disaggregation	+32
Knapsack cover	+64
Lattice	+128
Gomory	+256
Coefficient reduction	+512
GCD	+1024
Obj. integrality	+2048

Possible Values:

Any value.

Default value:

-1

- `mio_cut_level_tree`

Corresponding constant:

`mosek.iparam.mio_cut_level_tree`

Description:

Controls the cut level employed by the mixed-integer optimizer at the tree. See `mosek.iparam.mio_cut_level_ro` for an explanation of the parameter values.

Possible Values:

Any value.

Default value:

-1

- `mio_feaspump_level`

Corresponding constant:

`mosek.iparam.mio_feaspump_level`

Description:

Feasibility pump is a heuristic designed to compute an initial feasible solution. A value of 0 implies that the feasibility pump heuristic is not used. A value of -1 implies that the mixed-integer optimizer decides how the feasibility pump heuristic is used. A larger value than 1 implies that the feasibility pump is employed more aggressively. Normally a value beyond 3 is not worthwhile.

Possible Values:

Any number between -inf and 3.

Default value:

-1

- `mio_heuristic_level`

Corresponding constant:

`mosek.iparam.mio_heuristic_level`

Description:

Controls the heuristic employed by the mixed-integer optimizer to locate an initial good integer feasible solution. A value of zero means the heuristic is not used at all. A larger value than 0 means that a gradually more sophisticated heuristic is used which is computationally more expensive. A negative value implies that the optimizer chooses the heuristic. Normally a value around 3 to 5 should be optimal.

Possible Values:

Any value.

Default value:

-1

- `mio_hotstart`

Corresponding constant:

`mosek.iparam.mio_hotstart`

Description:

Controls whether the integer optimizer is hot-started.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.on`

- `mio_keep_basis`

Corresponding constant:

`mosek.iparam.mio_keep_basis`

Description:

Controls whether the integer presolve keeps bases in memory. This speeds on the solution process at cost of bigger memory consumption.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.on`

- `mio_local_branch_number`

Corresponding constant:

`mosek.iparam.mio_local_branch_number`

Description:

Controls the size of the local search space when doing local branching.

Possible Values:

Any number between -inf and +inf.

Default value:

-1

- `mio_max_num_branches`

Corresponding constant:

`mosek.iparam.mio_max_num_branches`

Description:

Maximum number of branches allowed during the branch and bound search. A negative value means infinite.

Possible Values:

Any number between -inf and +inf.

Default value:

-1

See also:

`mosek.dparam.mio_disable_term_time` Certain termination criteria is disabled within the mixed-integer optimizer for period time specified by the parameter.

- `mio_max_num_relaxs`

Corresponding constant:

`mosek.iparam.mio_max_num_relaxs`

Description:

Maximum number of relaxations allowed during the branch and bound search. A negative value means infinite.

Possible Values:

Any number between -inf and +inf.

Default value:

-1

See also:

`mosek.dparam.mio_disable_term_time` Certain termination criteria is disabled within the mixed-integer optimizer for period time specified by the parameter.

- `mio_max_num_solutions`

Corresponding constant:

`mosek.iparam.mio_max_num_solutions`

Description:

The mixed-integer optimizer can be terminated after a certain number of different feasible solutions has been located. If this parameter has the value n and n is strictly positive, then the mixed-integer optimizer will be terminated when n feasible solutions have been located.

Possible Values:

Any number between -inf and +inf.

Default value:

-1

See also:

`mosek.dparam.mio_disable_term_time` Certain termination criteria is disabled within the mixed-integer optimizer for period time specified by the parameter.

- `mio_mode`

Corresponding constant:

`mosek.iparam.mio_mode`

Description:

Controls whether the optimizer includes the integer restrictions when solving a (mixed) integer optimization problem.

Possible values:

`mosek.miomode.ignored` The integer constraints are ignored and the problem is solved as a continuous problem.

`mosek.miomode.lazy` Integer restrictions should be satisfied if an optimizer is available for the problem.

`mosek.miomode.satisfied` Integer restrictions should be satisfied.

Default value:

`mosek.miomode.satisfied`

- `mio_node_optimizer`

Corresponding constant:

`mosek.iparam.mio_node_optimizer`

Description:

Controls which optimizer is employed at the non-root nodes in the mixed-integer optimizer.

Possible values:

`mosek.optimizertype.intpnt` The interior-point optimizer is used.

`mosek.optimizertype.concurrent` The optimizer for nonconvex nonlinear problems.

`mosek.optimizertype.mixed_int` The mixed-integer optimizer.

`mosek.optimizertype.dual_simplex` The dual simplex optimizer is used.

`mosek.optimizertype.free` The optimizer is chosen automatically.

`mosek.optimizertype.primal_dual_simplex` The primal dual simplex optimizer is used.

`mosek.optimizertype.conic` The optimizer for problems having conic constraints.

`mosek.optimizertype.nonconvex` The optimizer for nonconvex nonlinear problems.

`mosek.optimizertype.qcone` For internal use only.

`mosek.optimizertype.primal_simplex` The primal simplex optimizer is used.

`mosek.optimizertype.free_simplex` One of the simplex optimizers is used.

Default value:

`mosek.optimizertype.free`

- `mio_node_selection`

Corresponding constant:

`mosek.iparam.mio_node_selection`

Description:

Controls the node selection strategy employed by the mixed-integer optimizer.

Possible values:

`mosek.mionodeseltype.pseudo` The optimizer employs selects the node based on a pseudo cost estimate.

`mosek.mionodeseltype.hybrid` The optimizer employs a hybrid strategy.

`mosek.mionodeseltype.free` The optimizer decides the node selection strategy.

`mosek.mionodeseltype.worst` The optimizer employs a worst bound node selection strategy.

`mosek.mionodeseltype.best` The optimizer employs a best bound node selection strategy.

`mosek.mionodeseltype.first` The optimizer employs a depth first node selection strategy.

Default value:

`mosek.mionodeseltype.free`

- `mio_optimizer_mode`

Corresponding constant:

`mosek.iparam.mio_optimizer_mode`

Description:

An experimental feature.

Possible Values:

Any number between 0 and 1.

Default value:

0

- `mio_presolve_aggregate`

Corresponding constant:

`mosek.iparam.mio_presolve_aggregate`

Description:

Controls whether the presolve used by the mixed-integer optimizer tries to aggregate the constraints.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.on`

- `mio_presolve_probing`

Corresponding constant:

`mosek.iparam.mio_presolve_probing`

Description:

Controls whether the mixed-integer presolve performs probing. Probing can be very time consuming.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.on`

- `mio_presolve_use`

Corresponding constant:

`mosek.iparam.mio_presolve_use`

Description:

Controls whether presolve is performed by the mixed-integer optimizer.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.on`

- `mio_root_optimizer`

Corresponding constant:

`mosek.iparam.mio_root_optimizer`

Description:

Controls which optimizer is employed at the root node in the mixed-integer optimizer.

Possible values:

`mosek.optimizertype.intpnt` The interior-point optimizer is used.

`mosek.optimizertype.concurrent` The optimizer for nonconvex nonlinear problems.

`mosek.optimizertype.mixed_int` The mixed-integer optimizer.

`mosek.optimizertype.dual_simplex` The dual simplex optimizer is used.

`mosek.optimizertype.free` The optimizer is chosen automatically.

`mosek.optimizertype.primal_dual_simplex` The primal dual simplex optimizer is used.

`mosek.optimizertype.conic` The optimizer for problems having conic constraints.

`mosek.optimizertype.nonconvex` The optimizer for nonconvex nonlinear problems.

`mosek.optimizertype.qcone` For internal use only.

`mosek.optimizertype.primal_simplex` The primal simplex optimizer is used.

`mosek.optimizertype.free_simplex` One of the simplex optimizers is used.

Default value:

`mosek.optimizertype.free`

- `mio_strong_branch`

Corresponding constant:

`mosek.iparam.mio_strong_branch`

Description:

The value specifies the depth from the root in which strong branching is used. A negative value means that the optimizer chooses a default value automatically.

Possible Values:

Any number between $-\infty$ and $+\infty$.

Default value:

-1

- `nonconvex_max_iterations`

Corresponding constant:

`mosek.iparam.nonconvex_max_iterations`

Description:

Maximum number of iterations that can be used by the nonconvex optimizer.

Possible Values:

Any number between 0 and $+\infty$.

Default value:

100000

- `objective_sense`

Corresponding constant:

`mosek.iparam.objective_sense`

Description:

If the objective sense for the task is undefined, then the value of this parameter is used as the default objective sense.

Possible values:

`mosek.objsense.minimize` The problem should be minimized.

`mosek.objsense.undefined` The objective sense is undefined.

`mosek.objsense.maximize` The problem should be maximized.

Default value:

`mosek.objsense.minimize`

- `opf_max_terms_per_line`

Corresponding constant:

`mosek.iparam.opf_max_terms_per_line`

Description:

The maximum number of terms (linear and quadratic) per line when an OPF file is written.

Possible Values:

Any number between 0 and +inf.

Default value:

5

- `opf_write_header`

Corresponding constant:

`mosek.iparam.opf_write_header`

Description:

Write a text header with date and MOSEK version in an OPF file.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.on`

- `opf_write_hints`

Corresponding constant:

`mosek.iparam.opf_write_hints`

Description:

Write a hint section with problem dimensions in the beginning of an OPF file.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.on`

- `opf_write_parameters`

Corresponding constant:

`mosek.iparam.opf_write_parameters`

Description:

Write a parameter section in an OPF file.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.off`

- `opf_write_problem`

Corresponding constant:

`mosek.iparam.opf_write_problem`

Description:

Write objective, constraints, bounds etc. to an OPF file.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.on`

- `opf_write_sol_bas`

Corresponding constant:

`mosek.iparam.opf_write_sol_bas`

Description:

If `mosek.iparam.opf_write_solutions` is `mosek.onoffkey.on` and a basic solution is defined, include the basic solution in OPF files.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.on`

- `opf_write_sol_itg`

Corresponding constant:

`mosek.iparam.opf_write_sol_itg`

Description:

If `mosek.iparam.opf_write_solutions` is `mosek.onoffkey.on` and an integer solution is defined, write the integer solution in OPF files.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.on`

- `opf_write_sol_itr`

Corresponding constant:

`mosek.iparam.opf_write_sol_itr`

Description:

If `mosek.iparam.opf_write_solutions` is `mosek.onoffkey.on` and an interior solution is defined, write the interior solution in OPF files.

Possible values:

mosek.onoffkey.on Switch the option on.
 mosek.onoffkey.off Switch the option off.

Default value:

mosek.onoffkey.on

- opf_write_solutions

Corresponding constant:

mosek.iparam.opf_write_solutions

Description:

Enable inclusion of solutions in the OPF files.

Possible values:

mosek.onoffkey.on Switch the option on.
 mosek.onoffkey.off Switch the option off.

Default value:

mosek.onoffkey.off

- optimizer

Corresponding constant:

mosek.iparam.optimizer

Description:

The paramter controls which optimizer is used to optimize the task.

Possible values:

mosek.optimizertype.intpnt The interior-point optimizer is used.
 mosek.optimizertype.concurrent The optimizer for nonconvex nonlinear problems.
 mosek.optimizertype.mixed_int The mixed-integer optimizer.
 mosek.optimizertype.dual_simplex The dual simplex optimizer is used.
 mosek.optimizertype.free The optimizer is chosen automatically.
 mosek.optimizertype.primal_dual_simplex The primal dual simplex optimizer is used.
 mosek.optimizertype.conic The optimizer for problems having conic constraints.
 mosek.optimizertype.nonconvex The optimizer for nonconvex nonlinear problems.
 mosek.optimizertype.qcone For internal use only.
 mosek.optimizertype.primal_simplex The primal simplex optimizer is used.
 mosek.optimizertype.free_simplex One of the simplex optimizers is used.

Default value:

mosek.optimizertype.free

- param_read_case_name

Corresponding constant:

mosek.iparam.param_read_case_name

Description:

If turned on, then names in the parameter file are case sensitive.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.on`

- `param_read_ign_error`

Corresponding constant:

`mosek.iparam.param_read_ign_error`

Description:

If turned on, then errors in parameter settings is ignored.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.off`

- `presolve_elim_fill`

Corresponding constant:

`mosek.iparam.presolve_elim_fill`

Description:

Controls the maximum amount of fill-in that can be created during the elimination phase of the presolve. This parameter times (`numcon+numvar`) denotes the amount of fill-in.

Possible Values:

Any number between 0 and `+inf`.

Default value:

1

- `presolve_eliminator_max_num_tries`

Corresponding constant:

`mosek.iparam.presolve_eliminator_max_num_tries`

Description:

Control the maximum number of times the eliminator is tried.

Possible Values:

A negative value implies MOSEK decides maximum number of times.

Default value:

-1

- `presolve_eliminator_use`

Corresponding constant:

`mosek.iparam.presolve_eliminator_use`

Description:

Controls whether free or implied free variables are eliminated from the problem.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.on`

- `presolve_level`

Corresponding constant:

`mosek.iparam.presolve_level`

Description:

Currently not used.

Possible Values:

Any number between $-\infty$ and $+\infty$.

Default value:

-1

- `presolve_lindep_use`

Corresponding constant:

`mosek.iparam.presolve_lindep_use`

Description:

Controls whether the linear constraints are checked for linear dependencies.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.on`

- `presolve_lindep_work_lim`

Corresponding constant:

`mosek.iparam.presolve_lindep_work_lim`

Description:

Is used to limit the amount of work that can be done to locate linear dependencies. In general the higher value this parameter is given the less work can be used. However, a value of 0 means no limit on the amount of work that can be used.

Possible Values:

Any number between 0 and $+\infty$.

Default value:

1

- `presolve_use`

Corresponding constant:

`mosek.iparam.presolve_use`

Description:

Controls whether the presolve is applied to a problem before it is optimized.

Possible values:

`mosek.presolvemode.on` The problem is presolved before it is optimized.

`mosek.presolvemode.off` The problem is not presolved before it is optimized.

`mosek.presolvemode.free` It is decided automatically whether to presolve before the problem is optimized.

Default value:

`mosek.presolvemode.free`

- `qo_separable_reformulation`

Corresponding constant:

`mosek.iparam.qo_separable_reformulation`

Description:

Determine if Quadratic programming problems should be reformulated to separable form.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.off`

- `read_add_anz`

Corresponding constant:

`mosek.iparam.read_add_anz`

Description:

Additional number of non-zeros in A that is made room for in the problem.

Possible Values:

Any number between 0 and $+\infty$.

Default value:

0

- `read_add_con`

Corresponding constant:

`mosek.iparam.read_add_con`

Description:

Additional number of constraints that is made room for in the problem.

Possible Values:

Any number between 0 and +inf.

Default value:

0

- `read_add_cone`

Corresponding constant:

`mosek.iparam.read_add_cone`

Description:

Additional number of conic constraints that is made room for in the problem.

Possible Values:

Any number between 0 and +inf.

Default value:

0

- `read_add_qnz`

Corresponding constant:

`mosek.iparam.read_add_qnz`

Description:

Additional number of non-zeros in the Q matrices that is made room for in the problem.

Possible Values:

Any number between 0 and +inf.

Default value:

0

- `read_add_var`

Corresponding constant:

`mosek.iparam.read_add_var`

Description:

Additional number of variables that is made room for in the problem.

Possible Values:

Any number between 0 and +inf.

Default value:

0

- `read_anz`

Corresponding constant:

`mosek.iparam.read_anz`

Description:

Expected maximum number of A non-zeros to be read. The option is used only by fast MPS and LP file readers.

Possible Values:

Any number between 0 and $+\infty$.

Default value:

100000

- `read_con`

Corresponding constant:

`mosek.iparam.read_con`

Description:

Expected maximum number of constraints to be read. The option is only used by fast MPS and LP file readers.

Possible Values:

Any number between 0 and $+\infty$.

Default value:

10000

- `read_cone`

Corresponding constant:

`mosek.iparam.read_cone`

Description:

Expected maximum number of conic constraints to be read. The option is used only by fast MPS and LP file readers.

Possible Values:

Any number between 0 and $+\infty$.

Default value:

2500

- `read_data_compressed`

Corresponding constant:

`mosek.iparam.read_data_compressed`

Description:

If this option is turned on, it is assumed that the data file is compressed.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.off`

- `read_data_format`

Corresponding constant:

`mosek.iparam.read_data_format`

Description:

Format of the data file to be read.

Possible values:

`mosek.dataformat.xml` The data file is an XML formatted file.
`mosek.dataformat.free_mps` The data data a free MPS formatted file.
`mosek.dataformat.extension` The file extension is used to determine the data file format.
`mosek.dataformat.mps` The data file is MPS formatted.
`mosek.dataformat.lp` The data file is LP formatted.
`mosek.dataformat.mbt` The data file is a MOSEK binary task file.
`mosek.dataformat.op` The data file is an optimization problem formatted file.

Default value:

`mosek.dataformat.extension`

- `read_keep_free_con`

Corresponding constant:

`mosek.iparam.read_keep_free_con`

Description:

Controls whether the free constraints are included in the problem.

Possible values:

`mosek.onoffkey.on` Switch the option on.
`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.off`

- `read_lp_drop_new_vars_in_bou`

Corresponding constant:

`mosek.iparam.read_lp_drop_new_vars_in_bou`

Description:

If this option is turned on, MOSEK will drop variables that are defined for the first time in the bounds section.

Possible values:

`mosek.onoffkey.on` Switch the option on.
`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.off`

- `read_lp_quoted_names`

Corresponding constant:

`mosek.iparam.read_lp_quoted_names`

Description:

If a name is in quotes when reading an LP file, the quotes will be removed.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.on`

- `read_mps_format`

Corresponding constant:

`mosek.iparam.read_mps_format`

Description:

Controls how strictly the MPS file reader interprets the MPS format.

Possible values:

`mosek.mpsformat.strict` It is assumed that the input file satisfies the MPS format strictly.

`mosek.mpsformat.relaxed` It is assumed that the input file satisfies a slightly relaxed version of the MPS format.

`mosek.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.

Default value:

`mosek.mpsformat.relaxed`

- `read_mps_keep_int`

Corresponding constant:

`mosek.iparam.read_mps_keep_int`

Description:

Controls whether MOSEK should keep the integer restrictions on the variables while reading the MPS file.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.on`

- `read_mps_obj_sense`

Corresponding constant:

`mosek.iparam.read_mps_obj_sense`

Description:

If turned on, the MPS reader uses the objective sense section. Otherwise the MPS reader ignores it.

Possible values:

mosek.onoffkey.on Switch the option on.
 mosek.onoffkey.off Switch the option off.

Default value:

mosek.onoffkey.on

- read_mps_quoted_names

Corresponding constant:

mosek.iparam.read_mps_quoted_names

Description:

If a name is in quotes when reading an MPS file, then the quotes will be removed.

Possible values:

mosek.onoffkey.on Switch the option on.
 mosek.onoffkey.off Switch the option off.

Default value:

mosek.onoffkey.on

- read_mps_relax

Corresponding constant:

mosek.iparam.read_mps_relax

Description:

If this option is turned on, then mixed integer constraints are ignored when a problem is read.

Possible values:

mosek.onoffkey.on Switch the option on.
 mosek.onoffkey.off Switch the option off.

Default value:

mosek.onoffkey.on

- read_mps_width

Corresponding constant:

mosek.iparam.read_mps_width

Description:

Controls the maximal number of characters allowed in one line of the MPS file.

Possible Values:

Any positive number greater than 80.

Default value:

1024

- read_q_mode

Corresponding constant:

`mosek.iparam.read_q_mode`

Description:

Controls how the Q matrices are read from the MPS file.

Possible values:

`mosek.qreadtype.add` All elements in a Q matrix are assumed to belong to the lower triangular part. Duplicate elements in a Q matrix are added together.

`mosek.qreadtype.drop_lower` All elements in the strict lower triangular part of the Q matrices are dropped.

`mosek.qreadtype.drop_upper` All elements in the strict upper triangular part of the Q matrices are dropped.

Default value:

`mosek.qreadtype.add`

- `read_qnz`

Corresponding constant:

`mosek.iparam.read_qnz`

Description:

Expected maximum number of Q non-zeros to be read. The option is used only by MPS and LP file readers.

Possible Values:

Any number between 0 and $+\infty$.

Default value:

20000

- `read_task_ignore_param`

Corresponding constant:

`mosek.iparam.read_task_ignore_param`

Description:

Controls whether MOSEK should ignore the parameter setting defined in the task file and use the default parameter setting instead.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.off`

- `read_var`

Corresponding constant:

`mosek.iparam.read_var`

Description:

Expected maximum number of variable to be read. The option is used only by MPS and LP file readers.

Possible Values:

Any number between 0 and +inf.

Default value:

10000

- `sensitivity_all`

Corresponding constant:

`mosek.iparam.sensitivity_all`

Description:

If set to `mosek.onoffkey.on`, then `Task.sensitivityreport` analyzes all bounds and variables instead of reading a specification from the file.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.off`

- `sensitivity_optimizer`

Corresponding constant:

`mosek.iparam.sensitivity_optimizer`

Description:

Controls which optimizer is used for optimal partition sensitivity analysis.

Possible values:

`mosek.optimizertype.intpnt` The interior-point optimizer is used.

`mosek.optimizertype.concurrent` The optimizer for nonconvex nonlinear problems.

`mosek.optimizertype.mixed_int` The mixed-integer optimizer.

`mosek.optimizertype.dual_simplex` The dual simplex optimizer is used.

`mosek.optimizertype.free` The optimizer is chosen automatically.

`mosek.optimizertype.primal_dual_simplex` The primal dual simplex optimizer is used.

`mosek.optimizertype.conic` The optimizer for problems having conic constraints.

`mosek.optimizertype.nonconvex` The optimizer for nonconvex nonlinear problems.

`mosek.optimizertype.qcone` For internal use only.

`mosek.optimizertype.primal_simplex` The primal simplex optimizer is used.

`mosek.optimizertype.free_simplex` One of the simplex optimizers is used.

Default value:

`mosek.optimizertype.free_simplex`

- `sensitivity_type`

Corresponding constant:

`mosek.iparam.sensitivity_type`

Description:

Controls which type of sensitivity analysis is to be performed.

Possible values:

`mosek.sensitivitytype.optimal_partition` Optimal partition sensitivity analysis is performed.

`mosek.sensitivitytype.basis` Basis sensitivity analysis is performed.

Default value:

`mosek.sensitivitytype.basis`

- `sim_basis_factor_use`

Corresponding constant:

`mosek.iparam.sim_basis_factor_use`

Description:

Controls whether a (LU) factorization of the basis is used in a hot-start. Forcing a refactorization sometimes improves the stability of the simplex optimizers, but in most cases there is a performance penalty.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.on`

- `sim_degen`

Corresponding constant:

`mosek.iparam.sim_degen`

Description:

Controls how aggressively degeneration is handled.

Possible values:

`mosek.simdegen.none` The simplex optimizer should use no degeneration strategy.

`mosek.simdegen.moderate` The simplex optimizer should use a moderate degeneration strategy.

`mosek.simdegen.minimum` The simplex optimizer should use a minimum degeneration strategy.

`mosek.simdegen.aggressive` The simplex optimizer should use an aggressive degeneration strategy.

`mosek.simdegen.free` The simplex optimizer chooses the degeneration strategy.

Default value:

mosek.simdegen.free

- `sim_dual_crash`

Corresponding constant:

mosek.iparam.sim_dual_crash

Description:

Controls whether crashing is performed in the dual simplex optimizer.

In general if a basis consists of more than $(100 - \text{this parameter value})\%$ fixed variables, then a crash will be performed.

Possible Values:

Any number between 0 and $+\infty$.

Default value:

90

- `sim_dual_phaseone_method`

Corresponding constant:

mosek.iparam.sim_dual_phaseone_method

Description:

An experimental feature.

Possible Values:

Any number between 0 and 10.

Default value:

0

- `sim_dual_restrict_selection`

Corresponding constant:

mosek.iparam.sim_dual_restrict_selection

Description:

The dual simplex optimizer can use a so-called restricted selection/pricing strategy to choose the outgoing variable. Hence, if restricted selection is applied, then the dual simplex optimizer first choose a subset of all the potential outgoing variables. Next, for some time it will choose the outgoing variable only among the subset. From time to time the subset is redefined.

A larger value of this parameter implies that the optimizer will be more aggressive in its restriction strategy, i.e. a value of 0 implies that the restriction strategy is not applied at all.

Possible Values:

Any number between 0 and 100.

Default value:

50

- `sim_dual_selection`

Corresponding constant:

`mosek.iparam.sim_dual_selection`

Description:

Controls the choice of the incoming variable, known as the selection strategy, in the dual simplex optimizer.

Possible values:

`mosek.simseltype.full` The optimizer uses full pricing.

`mosek.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.

`mosek.simseltype.free` The optimizer chooses the pricing strategy.

`mosek.simseltype.ase` The optimizer uses approximate steepest-edge pricing.

`mosek.simseltype.devex` The optimizer uses devex steepest-edge pricing (or if it is not available an approximate steep-edge selection).

`mosek.simseltype.se` The optimizer uses steepest-edge selection (or if it is not available an approximate steep-edge selection).

Default value:

`mosek.simseltype.free`

- `sim_exploit_dupvec`

Corresponding constant:

`mosek.iparam.sim_exploit_dupvec`

Description:

Controls if the simplex optimizers are allowed to exploit duplicated columns.

Possible values:

`mosek.simdupvec.on` Allow the simplex optimizer to exploit duplicated columns.

`mosek.simdupvec.off` Disallow the simplex optimizer to exploit duplicated columns.

`mosek.simdupvec.free` The simplex optimizer can choose freely.

Default value:

`mosek.simdupvec.off`

- `sim_hotstart`

Corresponding constant:

`mosek.iparam.sim_hotstart`

Description:

Controls the type of hot-start that the simplex optimizer perform.

Possible values:

`mosek.simhotstart.none` The simplex optimizer performs a coldstart.

`mosek.simhotstart.status_keys` Only the status keys of the constraints and variables are used to choose the type of hot-start.

`mosek.simhotstart.free` The simplex optimizer chooses the hot-start type.

Default value:

`mosek.simhotstart.free`

- `sim_hotstart_lu`

Corresponding constant:

`mosek.iparam.sim_hotstart_lu`

Description:

Determines if the simplex optimizer should exploit the initial factorization.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.on`

- `sim_integer`

Corresponding constant:

`mosek.iparam.sim_integer`

Description:

An experimental feature.

Possible Values:

Any number between 0 and 10.

Default value:

0

- `sim_max_iterations`

Corresponding constant:

`mosek.iparam.sim_max_iterations`

Description:

Maximum number of iterations that can be used by a simplex optimizer.

Possible Values:

Any number between 0 and +inf.

Default value:

10000000

- `sim_max_num_setbacks`

Corresponding constant:

`mosek.iparam.sim_max_num_setbacks`

Description:

Controls how many set-backs are allowed within a simplex optimizer. A set-back is an event where the optimizer moves in the wrong direction. This is impossible in theory but may happen due to numerical problems.

Possible Values:

Any number between 0 and +inf.

Default value:

250

- `sim_network.detect`

Corresponding constant:

`mosek.iparam.sim_network.detect`

Description:

The simplex optimizer is capable of exploiting a network flow component in a problem. However it is only worthwhile to exploit the network flow component if it is sufficiently large. This parameter controls how large the network component has to be in “relative” terms before it is exploited. For instance a value of 20 means at least 20% of the model should be a network before it is exploited. If this value is larger than 100 the network flow component is never detected or exploited.

Possible Values:

Any number between 0 and +inf.

Default value:

101

- `sim_network.detect_hotstart`

Corresponding constant:

`mosek.iparam.sim_network.detect_hotstart`

Description:

This parameter controls how large the network component in “relative” terms has to be before it is exploited in a simplex hot-start. The network component should be equal or larger than

`max(mosek.iparam.sim_network.detect, mosek.iparam.sim_network.detect_hotstart)`

before it is exploited. If this value is larger than 100 the network flow component is never detected or exploited.

Possible Values:

Any number between 0 and +inf.

Default value:

100

- `sim_network.detect_method`

Corresponding constant:

`mosek.iparam.sim_network.detect_method`

Description:

Controls which type of detection method the network extraction should use.

Possible values:

`mosek.networkdetect.simple` The network detection should use a very simple heuristic.
`mosek.networkdetect.advanced` The network detection should use a more advanced heuristic.
`mosek.networkdetect.free` The network detection is free.

Default value:

`mosek.networkdetect.free`

- `sim_non_singular`

Corresponding constant:

`mosek.iparam.sim_non_singular`

Description:

Controls if the simplex optimizer ensures a non-singular basis, if possible.

Possible values:

`mosek.onoffkey.on` Switch the option on.
`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.on`

- `sim_primal_crash`

Corresponding constant:

`mosek.iparam.sim_primal_crash`

Description:

Controls whether crashing is performed in the primal simplex optimizer.

In general, if a basis consists of more than $(100 - \text{this parameter value})\%$ fixed variables, then a crash will be performed.

Possible Values:

Any nonnegative integer value.

Default value:

90

- `sim_primal_phaseone_method`

Corresponding constant:

`mosek.iparam.sim_primal_phaseone_method`

Description:

An experimental feature.

Possible Values:

Any number between 0 and 10.

Default value:

0

- `sim_primal_restrict_selection`

Corresponding constant:

`mosek.iparam.sim_primal_restrict_selection`

Description:

The primal simplex optimizer can use a so-called restricted selection/pricing strategy to chooses the outgoing variable. Hence, if restricted selection is applied, then the primal simplex optimizer first choose a subset of all the potential incoming variables. Next, for some time it will choose the incoming variable only among the subset. From time to time the subset is redefined.

A larger value of this parameter implies that the optimizer will be more aggressive in its restriction strategy, i.e. a value of 0 implies that the restriction strategy is not applied at all.

Possible Values:

Any number between 0 and 100.

Default value:

50

- `sim_primal_selection`

Corresponding constant:

`mosek.iparam.sim_primal_selection`

Description:

Controls the choice of the incoming variable, known as the selection strategy, in the primal simplex optimizer.

Possible values:

`mosek.simseltype.full` The optimizer uses full pricing.

`mosek.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.

`mosek.simseltype.free` The optimizer chooses the pricing strategy.

`mosek.simseltype.ase` The optimizer uses approximate steepest-edge pricing.

`mosek.simseltype.devex` The optimizer uses devex steepest-edge pricing (or if it is not available an approximate steep-edge selection).

`mosek.simseltype.se` The optimizer uses steepest-edge selection (or if it is not available an approximate steep-edge selection).

Default value:

`mosek.simseltype.free`

- `sim_refactor_freq`

Corresponding constant:

`mosek.iparam.sim_refactor_freq`

Description:

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.

Possible Values:

Any number between 0 and $+\infty$.

Default value:

0

- `sim_reformulation`

Corresponding constant:

`mosek.iparam.sim_reformulation`

Description:

Controls if the simplex optimizers are allowed to reformulate the problem.

Possible values:

`mosek.simreform.on` Allow the simplex optimizer to reformulate the problem.

`mosek.simreform.aggressive` The simplex optimizer should use an aggressive reformulation strategy.

`mosek.simreform.off` Disallow the simplex optimizer to reformulate the problem.

`mosek.simreform.free` The simplex optimizer can choose freely.

Default value:

`mosek.simreform.off`

- `sim_save_lu`

Corresponding constant:

`mosek.iparam.sim_save_lu`

Description:

Controls if the LU factorization stored should be replaced with the LU factorization corresponding to the initial basis.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.off`

- `sim_scaling`

Corresponding constant:

`mosek.iparam.sim_scaling`

Description:

Controls how much effort is used in scaling the problem before a simplex optimizer is used.

Possible values:

`mosek.scalingtype.none` No scaling is performed.

`mosek.scalingtype.moderate` A conservative scaling is performed.

`mosek.scalingtype.aggressive` A very aggressive scaling is performed.

`mosek.scalingtype.free` The optimizer chooses the scaling heuristic.

Default value:

`mosek.scalingtype.free`

- `sim_scaling_method`

Corresponding constant:

`mosek.iparam.sim_scaling_method`

Description:

Controls how the problem is scaled before a simplex optimizer is used.

Possible values:

`mosek.scalingmethod.pow2` Scales only with power of 2 leaving the mantissa untouched.

`mosek.scalingmethod.free` The optimizer chooses the scaling heuristic.

Default value:

`mosek.scalingmethod.pow2`

- `sim_solve_form`

Corresponding constant:

`mosek.iparam.sim_solve_form`

Description:

Controls whether the primal or the dual problem is solved by the primal-/dual- simplex optimizer.

Possible values:

`mosek.solveform.primal` The optimizer should solve the primal problem.

`mosek.solveform.dual` The optimizer should solve the dual problem.

`mosek.solveform.free` The optimizer is free to solve either the primal or the dual problem.

Default value:

`mosek.solveform.free`

- `sim_stability_priority`

Corresponding constant:

`mosek.iparam.sim_stability_priority`

Description:

Controls how high priority the numerical stability should be given.

Possible Values:

Any number between 0 and 100.

Default value:

50

- `sim_switch_optimizer`

Corresponding constant:

`mosek.iparam.sim_switch_optimizer`

Description:

The simplex optimizer sometimes chooses to solve the dual problem instead of the primal problem. This implies that if you have chosen to use the dual simplex optimizer and the problem is dualized, then it actually makes sense to use the primal simplex optimizer instead. If this parameter is on and the problem is dualized and furthermore the simplex optimizer is chosen to be the primal (dual) one, then it is switched to the dual (primal).

Possible values:

`mosek.onoffkey.on` Switch the option on.
`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.off`

- `sol_filter_keep_basic`

Corresponding constant:

`mosek.iparam.sol_filter_keep_basic`

Description:

If turned on, then basic and super basic constraints and variables are written to the solution file independent of the filter setting.

Possible values:

`mosek.onoffkey.on` Switch the option on.
`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.off`

- `sol_filter_keep_ranged`

Corresponding constant:

`mosek.iparam.sol_filter_keep_ranged`

Description:

If turned on, then ranged constraints and variables are written to the solution file independent of the filter setting.

Possible values:

`mosek.onoffkey.on` Switch the option on.
`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.off`

- `sol_quoted_names`

Corresponding constant:

`mosek.iparam.sol_quoted_names`

Description:

If this options is turned on, then MOSEK will quote names that contains blanks while writing the solution file. Moreover when reading leading and trailing quotes will be stripped of.

Possible values:

`mosek.onoffkey.on` Switch the option on.
`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.off`

- `sol_read_name_width`

Corresponding constant:

`mosek.iparam.sol_read_name_width`

Description:

When a solution is read by MOSEK and some constraint, variable or cone names contain blanks, then a maximum name width much be specified. A negative value implies that no name contain blanks.

Possible Values:

Any number between `-inf` and `+inf`.

Default value:

`-1`

- `sol_read_width`

Corresponding constant:

`mosek.iparam.sol_read_width`

Description:

Controls the maximal acceptable width of line in the solutions when read by MOSEK.

Possible Values:

Any positive number greater than 80.

Default value:

`1024`

- `solution_callback`

Corresponding constant:

`mosek.iparam.solution_callback`

Description:

Indicates whether solution call-backs will be performed during the optimization.

Possible values:

`mosek.onoffkey.on` Switch the option on.
`mosek.onoffkey.off` Switch the option off.

Default value:`mosek.onoffkey.off`• **timing_level****Corresponding constant:**`mosek.iparam.timing_level`**Description:**

Controls the amount of timing performed inside MOSEK.

Possible Values:

Any integer greater or equal to 0.

Default value:

1

• **warning_level****Corresponding constant:**`mosek.iparam.warning_level`**Description:**

Warning level.

Possible Values:

Any number between 0 and +inf.

Default value:

1

• **write_bas_constraints****Corresponding constant:**`mosek.iparam.write_bas_constraints`**Description:**

Controls whether the constraint section is written to the basic solution file.

Possible values:`mosek.onoffkey.on` Switch the option on.`mosek.onoffkey.off` Switch the option off.**Default value:**`mosek.onoffkey.on`• **write_bas_head****Corresponding constant:**`mosek.iparam.write_bas_head`**Description:**

Controls whether the header section is written to the basic solution file.

Possible values:`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.on`

- `write_bas_variables`

Corresponding constant:

`mosek.iparam.write_bas_variables`

Description:

Controls whether the variables section is written to the basic solution file.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.on`

- `write_data_compressed`

Corresponding constant:

`mosek.iparam.write_data_compressed`

Description:

Controls whether the data file is compressed while it is written. 0 means no compression while higher values mean more compression.

Possible Values:

Any number between 0 and +inf.

Default value:

0

- `write_data_format`

Corresponding constant:

`mosek.iparam.write_data_format`

Description:

Controls the data format when a task is written using `Task.writedata`.

Possible values:

`mosek.dataformat.xml` The data file is an XML formatted file.

`mosek.dataformat.free_mps` The data data a free MPS formatted file.

`mosek.dataformat.extension` The file extension is used to determine the data file format.

`mosek.dataformat.mps` The data file is MPS formatted.

`mosek.dataformat.lp` The data file is LP formatted.

`mosek.dataformat.mbt` The data file is a MOSEK binary task file.

`mosek.dataformat.op` The data file is an optimization problem formatted file.

Default value:

`mosek.dataformat.extension`

- `write_data_param`

Corresponding constant:

`mosek.iparam.write_data_param`

Description:

If this option is turned on the parameter settings are written to the data file as parameters.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.off`

- `write_free_con`

Corresponding constant:

`mosek.iparam.write_free_con`

Description:

Controls whether the free constraints are written to the data file.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.off`

- `write_generic_names`

Corresponding constant:

`mosek.iparam.write_generic_names`

Description:

Controls whether the generic names or user-defined names are used in the data file.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.off`

- `write_generic_names_io`

Corresponding constant:

`mosek.iparam.write_generic_names_io`

Description:

Index origin used in generic names.

Possible Values:

Any number between 0 and +inf.

Default value:

1

- `write_int_constraints`

Corresponding constant:

`mosek.iparam.write_int_constraints`

Description:

Controls whether the constraint section is written to the integer solution file.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.on`

- `write_int_head`

Corresponding constant:

`mosek.iparam.write_int_head`

Description:

Controls whether the header section is written to the integer solution file.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.on`

- `write_int_variables`

Corresponding constant:

`mosek.iparam.write_int_variables`

Description:

Controls whether the variables section is written to the integer solution file.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.on`

- `write_lp_line_width`

Corresponding constant:

`mosek.iparam.write_lp_line_width`

Description:

Maximum width of line in an LP file written by MOSEK.

Possible Values:

Any positive number.

Default value:

80

- `write_lp_quoted_names`

Corresponding constant:

`mosek.iparam.write_lp_quoted_names`

Description:

If this option is turned on, then MOSEK will quote invalid LP names when writing an LP file.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.on`

- `write_lp_strict_format`

Corresponding constant:

`mosek.iparam.write_lp_strict_format`

Description:

Controls whether LP output files satisfy the LP format strictly.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.off`

- `write_lp_terms_per_line`

Corresponding constant:

`mosek.iparam.write_lp_terms_per_line`

Description:

Maximum number of terms on a single line in an LP file written by MOSEK. 0 means unlimited.

Possible Values:

Any number between 0 and +inf.

Default value:

10

- `write_mps_int`

Corresponding constant:

`mosek.iparam.write_mps_int`

Description:

Controls if marker records are written to the MPS file to indicate whether variables are integer restricted.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.on`

- `write_mps_obj_sense`

Corresponding constant:

`mosek.iparam.write_mps_obj_sense`

Description:

If turned off, the objective sense section is not written to the MPS file.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.on`

- `write_mps_quoted_names`

Corresponding constant:

`mosek.iparam.write_mps_quoted_names`

Description:

If a name contains spaces (blanks) when writing an MPS file, then the quotes will be removed.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.on`

- `write_mps_strict`

Corresponding constant:

`mosek.iparam.write_mps_strict`

Description:

Controls whether the written MPS file satisfies the MPS format strictly or not.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.off`

- `write_precision`

Corresponding constant:

`mosek.iparam.write_precision`

Description:

Controls the precision with which `double` numbers are printed in the MPS data file. In general it is not worthwhile to use a value higher than 15.

Possible Values:

Any number between 0 and `+inf`.

Default value:

8

- `write_sol_constraints`

Corresponding constant:

`mosek.iparam.write_sol_constraints`

Description:

Controls whether the constraint section is written to the solution file.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.on`

- `write_sol_head`

Corresponding constant:

`mosek.iparam.write_sol_head`

Description:

Controls whether the header section is written to the solution file.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.on`

- `write_sol_variables`

Corresponding constant:

`mosek.iparam.write_sol_variables`

Description:

Controls whether the variables section is written to the solution file.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.on`

- `write_task_inc_sol`

Corresponding constant:

`mosek.iparam.write_task_inc_sol`

Description:

Controls whether the solutions are stored in the task file too.

Possible values:

`mosek.onoffkey.on` Switch the option on.

`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.on`

- `write_xml_mode`

Corresponding constant:

`mosek.iparam.write_xml_mode`

Description:

Controls if linear coefficients should be written by row or column when writing in the XML file format.

Possible values:

`mosek.xmlwriteroutputtype.col` Write in column order.

`mosek.xmlwriteroutputtype.row` Write in row order.

Default value:

`mosek.xmlwriteroutputtype.row`

15.4 String parameter types

- `mosek.sparam.bas_sol_file_name` 378
Name of the bas solution file.
- `mosek.sparam.data_file_name` 378
Data are read and written to this file.
- `mosek.sparam.debug_file_name` 379
MOSEK debug file.
- `mosek.sparam.feasrepair_name_prefix` 379
Feasibility repair name prefix.
- `mosek.sparam.feasrepair_name_separator` 379
Feasibility repair name separator.
- `mosek.sparam.feasrepair_name_wsumviol` 379
Feasibility repair name violation name.
- `mosek.sparam.int_sol_file_name` 380
Name of the int solution file.
- `mosek.sparam.itr_sol_file_name` 380
Name of the itr solution file.
- `mosek.sparam.param_comment_sign` 380
Solution file comment character.
- `mosek.sparam.param_read_file_name` 380
Modifications to the parameter database is read from this file.
- `mosek.sparam.param_write_file_name` 381
The parameter database is written to this file.
- `mosek.sparam.read_mps_bou_name` 381
Name of the BOUNDS vector used. An empty name means that the first BOUNDS vector is used.
- `mosek.sparam.read_mps_obj_name` 381
Objective name in the MPS file.
- `mosek.sparam.read_mps_ran_name` 381
Name of the RANGE vector used. An empty name means that the first RANGE vector is used.
- `mosek.sparam.read_mps_rhs_name` 382
Name of the RHS used. An empty name means that the first RHS vector is used.
- `mosek.sparam.sensitivity_file_name` 382
Sensitivity report file name.

- `mosek.sparam.sensitivity_res_file_name` 382
Name of the sensitivity report output file.
- `mosek.sparam.sol_filter_xc_low` 382
Solution file filter.
- `mosek.sparam.sol_filter_xc_upr` 383
Solution file filter.
- `mosek.sparam.sol_filter_xx_low` 383
Solution file filter.
- `mosek.sparam.sol_filter_xx_upr` 383
Solution file filter.
- `mosek.sparam.stat_file_name` 384
Statistics file name.
- `mosek.sparam.stat_key` 384
Key used when writing the summary file.
- `mosek.sparam.stat_name` 384
Name used when writing the statistics file.
- `mosek.sparam.write_lp_gen_var_name` 384
Added variable names in the LP files.

- `bas_sol_file_name`

Corresponding constant:

`mosek.sparam.bas_sol_file_name`

Description:

Name of the `bas` solution file.

Possible Values:

Any valid file name.

Default value:

`""`

- `data_file_name`

Corresponding constant:

`mosek.sparam.data_file_name`

Description:

Data are read and written to this file.

Possible Values:

Any valid file name.

Default value:

`""`

- `debug_file_name`

Corresponding constant:`mosek.sparam.debug_file_name`**Description:**

MOSEK debug file.

Possible Values:

Any valid file name.

Default value:`""`

- `feasrepair_name_prefix`

Corresponding constant:`mosek.sparam.feasrepair_name_prefix`**Description:**

If the function `Task.relaxprimal` adds new constraints to the problem, then they are prefixed by the value of this parameter.

Possible Values:

Any valid string.

Default value:`"MSK-"`

- `feasrepair_name_separator`

Corresponding constant:`mosek.sparam.feasrepair_name_separator`**Description:**

Separator string for names of constraints and variables generated by `Task.relaxprimal`.

Possible Values:

Any valid string.

Default value:`"_"`

- `feasrepair_name_wsumviol`

Corresponding constant:`mosek.sparam.feasrepair_name_wsumviol`**Description:**

The constraint and variable associated with the total weighted sum of violations are each given the name of this parameter postfixed with `CON` and `VAR` respectively.

Possible Values:

Any valid string.

Default value:

"WSUMVIOL"

- `int_sol_file_name`

Corresponding constant:

`mosek.sparam.int_sol_file_name`

Description:

Name of the `int` solution file.

Possible Values:

Any valid file name.

Default value:

" "

- `itr_sol_file_name`

Corresponding constant:

`mosek.sparam.itr_sol_file_name`

Description:

Name of the `itr` solution file.

Possible Values:

Any valid file name.

Default value:

" "

- `param_comment_sign`

Corresponding constant:

`mosek.sparam.param_comment_sign`

Description:

Only the first character in this string is used. It is considered as a start of comment sign in the MOSEK parameter file. Spaces are ignored in the string.

Possible Values:

Any valid string.

Default value:

"%"

- `param_read_file_name`

Corresponding constant:

`mosek.sparam.param_read_file_name`

Description:

Modifications to the parameter database is read from this file.

Possible Values:

Any valid file name.

Default value:

""

- `param_write_file_name`

Corresponding constant:

`mosek.sparam.param_write_file_name`

Description:

The parameter database is written to this file.

Possible Values:

Any valid file name.

Default value:

""

- `read_mps_bou_name`

Corresponding constant:

`mosek.sparam.read_mps_bou_name`

Description:

Name of the BOUNDS vector used. An empty name means that the first BOUNDS vector is used.

Possible Values:

Any valid MPS name.

Default value:

""

- `read_mps_obj_name`

Corresponding constant:

`mosek.sparam.read_mps_obj_name`

Description:

Name of the free constraint used as objective function. An empty name means that the first constraint is used as objective function.

Possible Values:

Any valid MPS name.

Default value:

""

- `read_mps_ran_name`

Corresponding constant:

`mosek.sparam.read_mps_ran_name`

Description:

Name of the RANGE vector used. An empty name means that the first RANGE vector is used.

Possible Values:

Any valid MPS name.

Default value:

""

- `read_mps_rhs_name`

Corresponding constant:

`mosek.sparam.read_mps_rhs_name`

Description:

Name of the RHS used. An empty name means that the first RHS vector is used.

Possible Values:

Any valid MPS name.

Default value:

""

- `sensitivity_file_name`

Corresponding constant:

`mosek.sparam.sensitivity_file_name`

Description:

If defined `Task.sensitivityreport` reads this file as a sensitivity analysis data file specifying the type of analysis to be done.

Possible Values:

Any valid string.

Default value:

""

- `sensitivity_res_file_name`

Corresponding constant:

`mosek.sparam.sensitivity_res_file_name`

Description:

If this is a nonempty string, then `Task.sensitivityreport` writes results to this file.

Possible Values:

Any valid string.

Default value:

""

- `sol_filter_xc_low`

Corresponding constant:

`mosek.sparam.sol_filter_xc_low`

Description:

A filter used to determine which constraints should be listed in the solution file. A value of “0.5” means that all constraints having $xc[i] > 0.5$ should be listed, whereas “+0.5” means that all constraints having $xc[i] \geq blc[i] + 0.5$ should be listed. An empty filter means that no filter is applied.

Possible Values:

Any valid filter.

Default value:

""

- `sol_filter_xc_upr`

Corresponding constant:

`mosek.sparam.sol_filter_xc_upr`

Description:

A filter used to determine which constraints should be listed in the solution file. A value of “0.5” means that all constraints having $xc[i] < 0.5$ should be listed, whereas “-0.5” means all constraints having $xc[i] \leq buc[i] - 0.5$ should be listed. An empty filter means that no filter is applied.

Possible Values:

Any valid filter.

Default value:

""

- `sol_filter_xx_low`

Corresponding constant:

`mosek.sparam.sol_filter_xx_low`

Description:

A filter used to determine which variables should be listed in the solution file. A value of “0.5” means that all constraints having $xx[j] \geq 0.5$ should be listed, whereas “+0.5” means that all constraints having $xx[j] \geq blx[j] + 0.5$ should be listed. An empty filter means no filter is applied.

Possible Values:

Any valid filter..

Default value:

""

- `sol_filter_xx_upr`

Corresponding constant:

`mosek.sparam.sol_filter_xx_upr`

Description:

A filter used to determine which variables should be listed in the solution file. A value of “0.5” means that all constraints having $xx[j] < 0.5$ should be printed, whereas “-0.5” means all constraints having $xx[j] \leq bux[j] - 0.5$ should be listed. An empty filter means no filter is applied.

Possible Values:

Any valid file name.

Default value:

""

- `stat_file_name`

Corresponding constant:

`mosek.sparam.stat_file_name`

Description:

Statistics file name.

Possible Values:

Any valid file name.

Default value:

""

- `stat_key`

Corresponding constant:

`mosek.sparam.stat_key`

Description:

Key used when writing the summary file.

Possible Values:

Any valid XML string.

Default value:

""

- `stat_name`

Corresponding constant:

`mosek.sparam.stat_name`

Description:

Name used when writing the statistics file.

Possible Values:

Any valid XML string.

Default value:

""

- `write_lp_gen_var_name`

Corresponding constant:

`mosek.sparam.write_lp_gen_var_name`

Description:

Sometimes when an LP file is written additional variables must be inserted. They will have the prefix denoted by this parameter.

Possible Values:

Any valid string.

Default value:

"xmskgen"

Chapter 16

Response codes

- (0) `mosek.rescode.ok`
No error occurred.
- (50) `mosek.rescode.wrn_open_param_file`
The parameter file could not be opened.
- (51) `mosek.rescode.wrn_large_bound`
A numerically large bound value is specified.
- (52) `mosek.rescode.wrn_large_lo_bound`
A numerically large lower bound value is specified.
- (53) `mosek.rescode.wrn_large_up_bound`
A numerically large upper bound value is specified.
- (54) `mosek.rescode.wrn_large_con_fx`
An equality constraint is fixed to a numerically large value. This can cause numerical problems.
- (57) `mosek.rescode.wrn_large_cj`
A numerically large value is specified for one c_j .
- (62) `mosek.rescode.wrn_large_aij`
A numerically large value is specified for an $a_{i,j}$ element in A . The parameter `mosek.dparam.data_tol_aij_large` controls when an $a_{i,j}$ is considered large.
- (63) `mosek.rescode.wrn_zero_aij`
One or more zero elements are specified in A .
- (65) `mosek.rescode.wrn_name_max_len`
A name is longer than the buffer that is supposed to hold it.
- (66) `mosek.rescode.wrn_spar_max_len`
A value for a string parameter is longer than the buffer that is supposed to hold it.

- (70) `mosek.rescode.wrn_mps_split_rhs_vector`
An RHS vector is split into several nonadjacent parts in an MPS file.
- (71) `mosek.rescode.wrn_mps_split_ran_vector`
A RANGE vector is split into several nonadjacent parts in an MPS file.
- (72) `mosek.rescode.wrn_mps_split_bou_vector`
A BOUNDS vector is split into several nonadjacent parts in an MPS file.
- (80) `mosek.rescode.wrn_lp_old_quad_format`
Missing `'/2'` after quadratic expressions in bound or objective.
- (85) `mosek.rescode.wrn_lp_drop_variable`
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.
- (200) `mosek.rescode.wrn_nz_in_upr_tri`
Non-zero elements specified in the upper triangle of a matrix were ignored.
- (201) `mosek.rescode.wrn_dropped_nz_qobj`
One or more non-zero elements were dropped in the Q matrix in the objective.
- (250) `mosek.rescode.wrn_ignore_integer`
Ignored integer constraints.
- (251) `mosek.rescode.wrn_no_global_optimizer`
No global optimizer is available.
- (270) `mosek.rescode.wrn_mio_infeasible_final`
The final mixed-integer problem with all the integer variables fixed at their optimal values is infeasible.
- (300) `mosek.rescode.wrn_sol_filter`
Invalid solution filter is specified.
- (350) `mosek.rescode.wrn_undef_sol_file_name`
Undefined name occurred in a solution.
- (351) `mosek.rescode.wrn_sol_file_ignored_con`
One or more lines in the constraint section were ignored when reading a solution file.
- (352) `mosek.rescode.wrn_sol_file_ignored_var`
One or more lines in the variable section were ignored when reading a solution file.
- (400) `mosek.rescode.wrn_too_few_basis_vars`
An incomplete basis has been specified. Too few basis variables are specified.
- (405) `mosek.rescode.wrn_too_many_basis_vars`
A basis with too many variables has been specified.
- (500) `mosek.rescode.wrn_license_expire`
The license expires.

- (501) `mosek.rescode.wrn_license_server`
The license server is not responding.
- (502) `mosek.rescode.wrn_empty_name`
A variable or constraint name is empty. The output file may be invalid.
- (503) `mosek.rescode.wrn_using_generic_names`
The file writer reverts to generic names because a name is blank.
- (505) `mosek.rescode.wrn_license_feature_expire`
The license expires.
- (705) `mosek.rescode.wrn_zeros_in_sparse_row`
One or more (near) zero elements are specified in a sparse row of a matrix. It is redundant to specify zero elements. Hence it may indicate an error.
- (710) `mosek.rescode.wrn_zeros_in_sparse_col`
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.
- (800) `mosek.rescode.wrn_incomplete_linear_dependency_check`
The linear dependency check(s) was not completed and therefore the *A* matrix may contain linear dependencies.
- (801) `mosek.rescode.wrn_eliminator_space`
The eliminator is skipped at least once due to lack of space.
- (802) `mosek.rescode.wrn_presolve_outofspace`
The presolve is incomplete due to lack of space.
- (803) `mosek.rescode.wrn_presolve_bad_precision`
The presolve estimates that the model is specified with insufficient precision.
- (804) `mosek.rescode.wrn_write_discarded_cfix`
The fixed objective term could not be converted to a variable and was discarded in the output file.
- (805) `mosek.rescode.wrn_construct_solution_infeas`
After fixing the integer variables at the suggested values then the problem is infeasible.
- (807) `mosek.rescode.wrn_construct_invalid_sol_itg`
The initial value for one or more of the integer variables is not feasible.
- (810) `mosek.rescode.wrn_construct_no_sol_itg`
The construct solution requires an integer solution.
- (900) `mosek.rescode.wrn_ana_large_bounds`
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 $+\infty$ or $-\infty$.

- (901) `mosek.rescode.wrn_ana_c_zero`
This warning is issued by the problem analyzer, if the coefficients in the linear part of the objective are all zero.
- (902) `mosek.rescode.wrn_ana_empty_cols`
This warning is issued by the problem analyzer, if columns, in which all coefficients are zero, are found.
- (903) `mosek.rescode.wrn_ana_close_bounds`
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.
- (904) `mosek.rescode.wrn_ana_almost_int_bounds`
This warning is issued by the problem analyzer if a constraint is bound nearly integral.
- (1000) `mosek.rescode.err_license`
Invalid license.
- (1001) `mosek.rescode.err_license_expired`
The license has expired.
- (1002) `mosek.rescode.err_license_version`
The license is valid for another version of MOSEK.
- (1005) `mosek.rescode.err_size_license`
The problem is bigger than the license.
- (1006) `mosek.rescode.err_prob_license`
The software is not licensed to solve the problem.
- (1007) `mosek.rescode.err_file_license`
Invalid license file.
- (1008) `mosek.rescode.err_missing_license_file`
MOSEK cannot find the license file or license server. Usually this happens if the operating system variable `MOSEKLM_LICENSE_FILE` is not set up appropriately. Please see the MOSEK installation manual for details.
- (1010) `mosek.rescode.err_size_license_con`
The problem has too many constraints to be solved with the available license.
- (1011) `mosek.rescode.err_size_license_var`
The problem has too many variables to be solved with the available license.
- (1012) `mosek.rescode.err_size_license_intvar`
The problem contains too many integer variables to be solved with the available license.
- (1013) `mosek.rescode.err_optimizer_license`
The optimizer required is not licensed.

- (1014) `mosek.rescode.err_flexlm`
The FLEXlm license manager reported an error.
- (1015) `mosek.rescode.err_license_server`
The license server is not responding.
- (1016) `mosek.rescode.err_license_max`
Maximum number of licenses is reached.
- (1017) `mosek.rescode.err_license_moseklm_daemon`
The MOSEKLM license manager daemon is not up and running.
- (1018) `mosek.rescode.err_license_feature`
A requested feature is not available in the license file(s). Most likely due to an incorrect license system setup.
- (1019) `mosek.rescode.err_platform_not_licensed`
A requested license feature is not available for the required platform.
- (1020) `mosek.rescode.err_license_cannot_allocate`
The license system cannot allocate the memory required.
- (1021) `mosek.rescode.err_license_cannot_connect`
MOSEK cannot connect to the license server. Most likely the license server is not up and running.
- (1025) `mosek.rescode.err_license_invalid_hostid`
The host ID specified in the license file does not match the host ID of the computer.
- (1026) `mosek.rescode.err_license_server_version`
The version specified in the checkout request is greater than the highest version number the daemon supports.
- (1027) `mosek.rescode.err_license_no_server_support`
The license server does not support the requested feature. Possible reasons for this error include:
- The feature has expired.
 - The feature's start date is later than today's date.
 - The version requested is higher than feature's the highest supported version.
 - A corrupted license file.
- Try restarting the license and inspect the license server debug file, usually called `lmgrd.log`.
- (1030) `mosek.rescode.err_open_dl`
A dynamic link library could not be opened.
- (1035) `mosek.rescode.err_older_dll`
The dynamic link library is older than the specified version.
- (1036) `mosek.rescode.err_newer_dll`
The dynamic link library is newer than the specified version.

- (1040) `mosek.rescode.err_link_file_dll`
A file cannot be linked to a stream in the DLL version.
- (1045) `mosek.rescode.err_thread_mutex_init`
Could not initialize a mutex.
- (1046) `mosek.rescode.err_thread_mutex_lock`
Could not lock a mutex.
- (1047) `mosek.rescode.err_thread_mutex_unlock`
Could not unlock a mutex.
- (1048) `mosek.rescode.err_thread_create`
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.
- (1049) `mosek.rescode.err_thread_cond_init`
Could not initialize a condition.
- (1050) `mosek.rescode.err_unknown`
Unknown error.
- (1051) `mosek.rescode.err_space`
Out of space.
- (1052) `mosek.rescode.err_file_open`
Error while opening a file.
- (1053) `mosek.rescode.err_file_read`
File read error.
- (1054) `mosek.rescode.err_file_write`
File write error.
- (1055) `mosek.rescode.err_data_file_ext`
The data file format cannot be determined from the file name.
- (1056) `mosek.rescode.err_invalid_file_name`
An invalid file name has been specified.
- (1057) `mosek.rescode.err_invalid_sol_file_name`
An invalid file name has been specified.
- (1058) `mosek.rescode.err_invalid_mbt_file`
A MOSEK binary task file is invalid.
- (1059) `mosek.rescode.err_end_of_file`
End of file reached.
- (1060) `mosek.rescode.err_null_env`
`env` is a NULL pointer.

- (1061) `mosek.rescode.err_null_task`
task is a NULL pointer.
- (1062) `mosek.rescode.err_invalid_stream`
An invalid stream is referenced.
- (1063) `mosek.rescode.err_no_init_env`
env is not initialized.
- (1064) `mosek.rescode.err_invalid_task`
The task is invalid.
- (1065) `mosek.rescode.err_null_pointer`
An argument to a function is unexpectedly a NULL pointer.
- (1066) `mosek.rescode.err_living_tasks`
All tasks associated with an environment must be deleted before the environment is deleted. There are still some undeleted tasks.
- (1070) `mosek.rescode.err_blank_name`
An all blank name has been specified.
- (1071) `mosek.rescode.err_dup_name`
The same name was used multiple times for the same problem item type.
- (1075) `mosek.rescode.err_invalid_obj_name`
An invalid objective name is specified.
- (1080) `mosek.rescode.err_space_leaking`
MOSEK is leaking memory. This can be due to either an incorrect use of MOSEK or a bug.
- (1081) `mosek.rescode.err_space_no_info`
No available information about the space usage.
- (1090) `mosek.rescode.err_read_format`
The specified format cannot be read.
- (1100) `mosek.rescode.err_mps_file`
An error occurred while reading an MPS file.
- (1101) `mosek.rescode.err_mps_inv_field`
A field in the MPS file is invalid. Probably it is too wide.
- (1102) `mosek.rescode.err_mps_inv_marker`
An invalid marker has been specified in the MPS file.
- (1103) `mosek.rescode.err_mps_null_con_name`
An empty constraint name is used in an MPS file.
- (1104) `mosek.rescode.err_mps_null_var_name`
An empty variable name is used in an MPS file.

- (1105) `mosek.rescode.err_mps_undef_con_name`
An undefined constraint name occurred in an MPS file.
- (1106) `mosek.rescode.err_mps_undef_var_name`
An undefined variable name occurred in an MPS file.
- (1107) `mosek.rescode.err_mps_inv_con_key`
An invalid constraint key occurred in an MPS file.
- (1108) `mosek.rescode.err_mps_inv_bound_key`
An invalid bound key occurred in an MPS file.
- (1109) `mosek.rescode.err_mps_inv_sec_name`
An invalid section name occurred in an MPS file.
- (1110) `mosek.rescode.err_mps_no_objective`
No objective is defined in an MPS file.
- (1111) `mosek.rescode.err_mps splitted_var`
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.
- (1112) `mosek.rescode.err_mps_mul_con_name`
A constraint name was specified multiple times in the `ROWS` section.
- (1113) `mosek.rescode.err_mps_mul_qsec`
Multiple `QSECTION`s are specified for a constraint in the MPS data file.
- (1114) `mosek.rescode.err_mps_mul_qobj`
The `Q` term in the objective is specified multiple times in the MPS data file.
- (1115) `mosek.rescode.err_mps_inv_sec_order`
The sections in the MPS data file are not in the correct order.
- (1116) `mosek.rescode.err_mps_mul_csec`
Multiple `CSECTION`s are given the same name.
- (1117) `mosek.rescode.err_mps_cone_type`
Invalid cone type specified in a `CSECTION`.
- (1118) `mosek.rescode.err_mps_cone_overlap`
A variable is specified to be a member of several cones.
- (1119) `mosek.rescode.err_mps_cone_repeat`
A variable is repeated within the `CSECTION`.
- (1122) `mosek.rescode.err_mps_invalid_objsense`
An invalid objective sense is specified.
- (1125) `mosek.rescode.err_mps_tab_in_field2`
A tab char occurred in field 2.

- (1126) `mosek.rescode.err_mps_tab_in_field3`
A tab char occurred in field 3.
- (1127) `mosek.rescode.err_mps_tab_in_field5`
A tab char occurred in field 5.
- (1128) `mosek.rescode.err_mps_invalid_obj_name`
An invalid objective name is specified.
- (1130) `mosek.rescode.err_ord_invalid_branch_dir`
An invalid branch direction key is specified.
- (1131) `mosek.rescode.err_ord_invalid`
Invalid content in branch ordering file.
- (1150) `mosek.rescode.err_lp_incompatible`
The problem cannot be written to an LP formatted file.
- (1151) `mosek.rescode.err_lp_empty`
The problem cannot be written to an LP formatted file.
- (1152) `mosek.rescode.err_lp_dup_slack_name`
The name of the slack variable added to a ranged constraint already exists.
- (1153) `mosek.rescode.err_write_mps_invalid_name`
An invalid name is created while writing an MPS file. Usually this will make the MPS file unreadable.
- (1154) `mosek.rescode.err_lp_invalid_var_name`
A variable name is invalid when used in an LP formatted file.
- (1155) `mosek.rescode.err_lp_free_constraint`
Free constraints cannot be written in LP file format.
- (1156) `mosek.rescode.err_write_opf_invalid_var_name`
Empty variable names cannot be written to OPF files.
- (1157) `mosek.rescode.err_lp_file_format`
Syntax error in an LP file.
- (1158) `mosek.rescode.err_write_lp_format`
Problem cannot be written as an LP file.
- (1159) `mosek.rescode.err_read_lp_missing_end_tag`
Missing End tag in LP file.
- (1160) `mosek.rescode.err_lp_format`
Syntax error in an LP file.
- (1161) `mosek.rescode.err_write_lp_non_unique_name`
An auto-generated name is not unique.

- (1162) `mosek.rescode.err_read_lp_nonexisting_name`
A variable never occurred in objective or constraints.
- (1163) `mosek.rescode.err_lp_write_conic_problem`
The problem contains cones that cannot be written to an LP formatted file.
- (1164) `mosek.rescode.err_lp_write_geco_problem`
The problem contains general convex terms that cannot be written to an LP formatted file.
- (1166) `mosek.rescode.err_writing_file`
An error occurred while writing file
- (1168) `mosek.rescode.err_opf_format`
Syntax error in an OPF file
- (1169) `mosek.rescode.err_opf_new_variable`
Introducing new variables is now allowed. When a `[variables]` section is present, it is not allowed to introduce new variables later in the problem.
- (1170) `mosek.rescode.err_invalid_name_in_sol_file`
An invalid name occurred in a solution file.
- (1171) `mosek.rescode.err_lp_invalid_con_name`
A constraint name is invalid when used in an LP formatted file.
- (1172) `mosek.rescode.err_opf_premature_eof`
Premature end of file in an OPF file.
- (1197) `mosek.rescode.err_argument_lenneq`
Incorrect length of arguments.
- (1198) `mosek.rescode.err_argument_type`
Incorrect argument type.
- (1199) `mosek.rescode.err_nr_arguments`
Incorrect number of function arguments.
- (1200) `mosek.rescode.err_in_argument`
A function argument is incorrect.
- (1201) `mosek.rescode.err_argument_dimension`
A function argument is of incorrect dimension.
- (1203) `mosek.rescode.err_index_is_too_small`
An index in an argument is too small.
- (1204) `mosek.rescode.err_index_is_too_large`
An index in an argument is too large.
- (1205) `mosek.rescode.err_param_name`
The parameter name is not correct.

- (1206) `mosek.rescode.err_param_name_dou`
The parameter name is not correct for a double parameter.
- (1207) `mosek.rescode.err_param_name_int`
The parameter name is not correct for an integer parameter.
- (1208) `mosek.rescode.err_param_name_str`
The parameter name is not correct for a string parameter.
- (1210) `mosek.rescode.err_param_index`
Parameter index is out of range.
- (1215) `mosek.rescode.err_param_is_too_large`
The parameter value is too large.
- (1216) `mosek.rescode.err_param_is_too_small`
The parameter value is too small.
- (1217) `mosek.rescode.err_param_value_str`
The parameter value string is incorrect.
- (1218) `mosek.rescode.err_param_type`
The parameter type is invalid.
- (1219) `mosek.rescode.err_inf_dou_index`
A double information index is out of range for the specified type.
- (1220) `mosek.rescode.err_inf_int_index`
An integer information index is out of range for the specified type.
- (1221) `mosek.rescode.err_index_arr_is_too_small`
An index in an array argument is too small.
- (1222) `mosek.rescode.err_index_arr_is_too_large`
An index in an array argument is too large.
- (1225) `mosek.rescode.err_inf_lint_index`
A long integer information index is out of range for the specified type.
- (1230) `mosek.rescode.err_inf_dou_name`
A double information name is invalid.
- (1231) `mosek.rescode.err_inf_int_name`
An integer information name is invalid.
- (1232) `mosek.rescode.err_inf_type`
The information type is invalid.
- (1234) `mosek.rescode.err_inf_lint_name`
A long integer information name is invalid.
- (1235) `mosek.rescode.err_index`
An index is out of range.

- (1236) `mosek.rescode.err_whichsol`
The solution defined by `compwhichsol` does not exists.
- (1237) `mosek.rescode.err_solitem`
The solution item number `solitem` is invalid. Please note that `mosek.solitem.snx` is invalid for the basic solution.
- (1238) `mosek.rescode.err_whichitem_not_allowed`
`whichitem` is unacceptable.
- (1240) `mosek.rescode.err_maxnumcon`
The maximum number of constraints specified is smaller than the number of constraints in the task.
- (1241) `mosek.rescode.err_maxnumvar`
The maximum number of variables specified is smaller than the number of variables in the task.
- (1243) `mosek.rescode.err_maxnumqnz`
The maximum number of non-zeros specified for the Q matrices is smaller than the number of non-zeros in the current Q matrices.
- (1250) `mosek.rescode.err_numconlim`
Maximum number of constraints limit is exceeded.
- (1251) `mosek.rescode.err_numvarlim`
Maximum number of variables limit is exceeded.
- (1252) `mosek.rescode.err_too_small_maxnumanz`
The maximum number of non-zeros specified for A is smaller than the number of non-zeros in the current A .
- (1253) `mosek.rescode.err_inv_aptre`
`aptre[j]` is strictly smaller than `aptrb[j]` for some j .
- (1254) `mosek.rescode.err_mul_a_element`
An element in A is defined multiple times.
- (1255) `mosek.rescode.err_inv_bk`
Invalid bound key.
- (1256) `mosek.rescode.err_inv_bkc`
Invalid bound key is specified for a constraint.
- (1257) `mosek.rescode.err_inv_bkx`
An invalid bound key is specified for a variable.
- (1258) `mosek.rescode.err_inv_var_type`
An invalid variable type is specified for a variable.
- (1259) `mosek.rescode.err_solver_probtype`
Problem type does not match the chosen optimizer.

- (1260) `mosek.rescode.err_objective_range`
Empty objective range.
- (1261) `mosek.rescode.err_first`
Invalid `first`.
- (1262) `mosek.rescode.err_last`
Invalid index `last`. A given index was out of expected range.
- (1263) `mosek.rescode.err_negative_surplus`
Negative surplus.
- (1264) `mosek.rescode.err_negative_append`
Cannot append a negative number.
- (1265) `mosek.rescode.err_undef_solution`
MOSEK has the following solution types:
 - an interior-point solution,
 - an 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.

- (1266) `mosek.rescode.err_basis`
An invalid basis is specified. Either too many or too few basis variables are specified.
- (1267) `mosek.rescode.err_inv_sk`
Invalid value in `sk`.
- (1268) `mosek.rescode.err_inv_skx`
Invalid value in `skx`.
- (1269) `mosek.rescode.err_inv_sk_str`
Invalid status key string encountered.
- (1270) `mosek.rescode.err_inv_sk`
Invalid status key code.
- (1271) `mosek.rescode.err_inv_cone_type_str`
Invalid cone type string encountered.
- (1272) `mosek.rescode.err_inv_cone_type`
Invalid cone type code is encountered.
- (1274) `mosek.rescode.err_inv_skn`
Invalid value in `skn`.

- (1275) `mosek.rescode.err_invalid_surplus`
Invalid surplus.
- (1280) `mosek.rescode.err_inv_name_item`
An invalid name item code is used.
- (1281) `mosek.rescode.err_pro_item`
An invalid problem is used.
- (1283) `mosek.rescode.err_invalid_format_type`
Invalid format type.
- (1285) `mosek.rescode.err_firsti`
Invalid `firsti`.
- (1286) `mosek.rescode.err_lasti`
Invalid `lasti`.
- (1287) `mosek.rescode.err_firstj`
Invalid `firstj`.
- (1288) `mosek.rescode.err_lastj`
Invalid `lastj`.
- (1290) `mosek.rescode.err_nonlinear_equality`
The model contains a nonlinear equality which defines a nonconvex set.
- (1291) `mosek.rescode.err_nonconvex`
The optimization problem is nonconvex.
- (1292) `mosek.rescode.err_nonlinear_ranged`
The model contains a nonlinear ranged constraint which by definition defines a nonconvex set.
- (1293) `mosek.rescode.err_con_q_not_psd`
The quadratic constraint matrix is not positive semi-definite as expected for a constraint with finite upper bound. This results in a nonconvex problem.
- (1294) `mosek.rescode.err_con_q_not_nsd`
The quadratic constraint matrix is not negative semi-definite as expected for a constraint with finite lower bound. This results in a nonconvex problem.
- (1295) `mosek.rescode.err_obj_q_not_psd`
The quadratic coefficient matrix in the objective is not positive semi-definite as expected for a minimization problem.
- (1296) `mosek.rescode.err_obj_q_not_nsd`
The quadratic coefficient matrix in the objective is not negative semi-definite as expected for a maximization problem.
- (1299) `mosek.rescode.err_argument_perm_array`
An invalid permutation array is specified.

- (1300) `mosek.rescode.err_cone_index`
An index of a non-existing cone has been specified.
- (1301) `mosek.rescode.err_cone_size`
A cone with too few members is specified.
- (1302) `mosek.rescode.err_cone_overlap`
A new cone which variables overlap with an existing cone has been specified.
- (1303) `mosek.rescode.err_cone_rep_var`
A variable is included multiple times in the cone.
- (1304) `mosek.rescode.err_maxnumcone`
The value specified for `maxnumcone` is too small.
- (1305) `mosek.rescode.err_cone_type`
Invalid cone type specified.
- (1306) `mosek.rescode.err_cone_type_str`
Invalid cone type specified.
- (1310) `mosek.rescode.err_remove_cone_variable`
A variable cannot be removed because it will make a cone invalid.
- (1350) `mosek.rescode.err_sol_file_invalid_number`
An invalid number is specified in a solution file.
- (1375) `mosek.rescode.err_huge_c`
A huge value in absolute size is specified for one c_j .
- (1380) `mosek.rescode.err_huge_aij`
A numerically huge value is specified for an $a_{i,j}$ element in A . The parameter `mosek.dparam.data_tol_aij_huge` controls when an $a_{i,j}$ is considered huge.
- (1400) `mosek.rescode.err_infinite_bound`
A numerically huge bound value is specified.
- (1401) `mosek.rescode.err_inv_qobj_subi`
Invalid value in `qosubi`.
- (1402) `mosek.rescode.err_inv_qobj_subj`
Invalid value in `qosubj`.
- (1403) `mosek.rescode.err_inv_qobj_val`
Invalid value in `qoval`.
- (1404) `mosek.rescode.err_inv_qcon_subk`
Invalid value in `qcsubk`.
- (1405) `mosek.rescode.err_inv_qcon_subi`
Invalid value in `qcsubi`.

- (1406) `mosek.rescode.err_inv_qcon_subj`
Invalid value in `qcsbj`.
- (1407) `mosek.rescode.err_inv_qcon_val`
Invalid value in `qcval`.
- (1408) `mosek.rescode.err_qcon_subi_too_small`
Invalid value in `qcsubi`.
- (1409) `mosek.rescode.err_qcon_subi_too_large`
Invalid value in `qcsubi`.
- (1415) `mosek.rescode.err_qobj_upper_triangle`
An element in the upper triangle of Q^o is specified. Only elements in the lower triangle should be specified.
- (1417) `mosek.rescode.err_qcon_upper_triangle`
An element in the upper triangle of a Q^k is specified. Only elements in the lower triangle should be specified.
- (1425) `mosek.rescode.err_fixed_bound_values`
A fixed constraint/variable has been specified using the bound keys but the numerical value of the lower and upper bound is different.
- (1430) `mosek.rescode.err_user_func_ret`
An user function reported an error.
- (1431) `mosek.rescode.err_user_func_ret_data`
An user function returned invalid data.
- (1432) `mosek.rescode.err_user_nlo_func`
The user-defined nonlinear function reported an error.
- (1433) `mosek.rescode.err_user_nlo_eval`
The user-defined nonlinear function reported an error.
- (1440) `mosek.rescode.err_user_nlo_eval_hessubi`
The user-defined nonlinear function reported an invalid subscript in the Hessian.
- (1441) `mosek.rescode.err_user_nlo_eval_hessubj`
The user-defined nonlinear function reported an invalid subscript in the Hessian.
- (1445) `mosek.rescode.err_invalid_objective_sense`
An invalid objective sense is specified.
- (1446) `mosek.rescode.err_undefined_objective_sense`
The objective sense has not been specified before the optimization.
- (1449) `mosek.rescode.err_y_is_undefined`
The solution item y is undefined.

- (1450) `mosek.rescode.err_nan_in_double_data`
An invalid floating point value was used in some double data.
- (1461) `mosek.rescode.err_nan_in_blc`
 l^c contains an invalid floating point value, i.e. a NaN.
- (1462) `mosek.rescode.err_nan_in_buc`
 u^c contains an invalid floating point value, i.e. a NaN.
- (1470) `mosek.rescode.err_nan_in_c`
 c contains an invalid floating point value, i.e. a NaN.
- (1471) `mosek.rescode.err_nan_in_blx`
 l^x contains an invalid floating point value, i.e. a NaN.
- (1472) `mosek.rescode.err_nan_in_bux`
 u^x contains an invalid floating point value, i.e. a NaN.
- (1473) `mosek.rescode.err_nan_in_aij`
 $a_{i,j}$ contains an invalid floating point value, i.e. a NaN.
- (1500) `mosek.rescode.err_inv_problem`
Invalid problem type. Probably a nonconvex problem has been specified.
- (1501) `mosek.rescode.err_mixed_problem`
The problem contains both conic and nonlinear constraints.
- (1550) `mosek.rescode.err_inv_optimizer`
An invalid optimizer has been chosen for the problem. This means that the simplex or the conic optimizer is chosen to optimize a nonlinear problem.
- (1551) `mosek.rescode.err_mio_no_optimizer`
No optimizer is available for the current class of integer optimization problems.
- (1552) `mosek.rescode.err_no_optimizer_var_type`
No optimizer is available for this class of optimization problems.
- (1553) `mosek.rescode.err_mio_not_loaded`
The mixed-integer optimizer is not loaded.
- (1580) `mosek.rescode.err_postsolve`
An error occurred during the postsolve. Please contact MOSEK support.
- (1590) `mosek.rescode.err_overflow`
A computation produced an overflow i.e. a very large number.
- (1600) `mosek.rescode.err_no_basis_sol`
No basic solution is defined.
- (1610) `mosek.rescode.err_basis_factor`
The factorization of the basis is invalid.

- (1615) `mosek.rescode.err_basis_singular`
The basis is singular and hence cannot be factored.
- (1650) `mosek.rescode.err_factor`
An error occurred while factorizing a matrix.
- (1700) `mosek.rescode.err_feasrepair_cannot_relax`
An optimization problem cannot be relaxed. This is the case e.g. for general nonlinear optimization problems.
- (1701) `mosek.rescode.err_feasrepair_solving_relaxed`
The relaxed problem could not be solved to optimality. Please consult the log file for further details.
- (1702) `mosek.rescode.err_feasrepair_inconsistent_bound`
The upper bound is less than the lower bound for a variable or a constraint. Please correct this before running the feasibility repair.
- (1750) `mosek.rescode.err_name_max_len`
A name is longer than the buffer that is supposed to hold it.
- (1760) `mosek.rescode.err_name_is_null`
The name buffer is a NULL pointer.
- (1800) `mosek.rescode.err_invalid_compression`
Invalid compression type.
- (1801) `mosek.rescode.err_invalid_iomode`
Invalid io mode.
- (2000) `mosek.rescode.err_no_primal_infeas_cer`
A certificate of primal infeasibility is not available.
- (2001) `mosek.rescode.err_no_dual_infeas_cer`
A certificate of infeasibility is not available.
- (2500) `mosek.rescode.err_no_solution_in_callback`
The required solution is not available.
- (2501) `mosek.rescode.err_inv_mark_i`
Invalid value in marki.
- (2502) `mosek.rescode.err_inv_mark_j`
Invalid value in markj.
- (2503) `mosek.rescode.err_inv_num_i`
Invalid numi.
- (2504) `mosek.rescode.err_inv_num_j`
Invalid numj.

- (2505) `mosek.rescode.err_cannot_clone_nl`
A task with a nonlinear function call-back cannot be cloned.
- (2506) `mosek.rescode.err_cannot_handle_nl`
A function cannot handle a task with nonlinear function call-backs.
- (2520) `mosek.rescode.err_invalid_accmode`
An invalid access mode is specified.
- (2550) `mosek.rescode.err_mbt_incompatible`
The MBT file is incompatible with this platform. This results from reading a file on a 32 bit platform generated on a 64 bit platform.
- (2800) `mosek.rescode.err_lu_max_num_tries`
Could not compute the LU factors of the matrix within the maximum number of allowed tries.
- (2900) `mosek.rescode.err_invalid_utf8`
An invalid UTF8 string is encountered.
- (2901) `mosek.rescode.err_invalid_wchar`
An invalid wchar string is encountered.
- (2950) `mosek.rescode.err_no_dual_for_itg_sol`
No dual information is available for the integer solution.
- (3000) `mosek.rescode.err_internal`
An internal error occurred. Please report this problem.
- (3001) `mosek.rescode.err_api_array_too_small`
An input array was too short.
- (3002) `mosek.rescode.err_api_cb_connect`
Failed to connect a callback object.
- (3005) `mosek.rescode.err_api_fatal_error`
An internal error occurred in the API. Please report this problem.
- (3050) `mosek.rescode.err_sen_format`
Syntax error in sensitivity analysis file.
- (3051) `mosek.rescode.err_sen_undef_name`
An undefined name was encountered in the sensitivity analysis file.
- (3052) `mosek.rescode.err_sen_index_range`
Index out of range in the sensitivity analysis file.
- (3053) `mosek.rescode.err_sen_bound_invalid_up`
Analysis of upper bound requested for an index, where no upper bound exists.
- (3054) `mosek.rescode.err_sen_bound_invalid_lo`
Analysis of lower bound requested for an index, where no lower bound exists.

- (3055) `mosek.rescode.err_sen_index_invalid`
Invalid range given in the sensitivity file.
- (3056) `mosek.rescode.err_sen_invalid_regexp`
Syntax error in regexp or regexp longer than 1024.
- (3057) `mosek.rescode.err_sen_solution_status`
No optimal solution found to the original problem given for sensitivity analysis.
- (3058) `mosek.rescode.err_sen_numerical`
Numerical difficulties encountered performing the sensitivity analysis.
- (3059) `mosek.rescode.err_concurrent_optimizer`
An unsupported optimizer was chosen for use with the concurrent optimizer.
- (3100) `mosek.rescode.err_unb_step_size`
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.
- (3101) `mosek.rescode.err_identical_tasks`
Some tasks related to this function call were identical. Unique tasks were expected.
- (3102) `mosek.rescode.err_ad_invalid_codelist`
The code list data was invalid.
- (3103) `mosek.rescode.err_ad_invalid_operator`
The code list data was invalid. An unknown operator was used.
- (3104) `mosek.rescode.err_ad_invalid_operand`
The code list data was invalid. An unknown operand was used.
- (3105) `mosek.rescode.err_ad_missing_operand`
The code list data was invalid. Missing operand for operator.
- (3106) `mosek.rescode.err_ad_missing_return`
The code list data was invalid. Missing return operation in function.
- (3200) `mosek.rescode.err_invalid_branch_direction`
An invalid branching direction is specified.
- (3201) `mosek.rescode.err_invalid_branch_priority`
An invalid branching priority is specified. It should be nonnegative.
- (3500) `mosek.rescode.err_internal_test_failed`
An internal unit test function failed.
- (3600) `mosek.rescode.err_xml_invalid_problem_type`
The problem type is not supported by the XML format.
- (3700) `mosek.rescode.err_invalid_ampl_stub`
Invalid AMPL stub.

- (3800) `mosek.rescode.err_int64_to_int32_cast`
An 32 bit integer could not cast to a 64 bit integer.
- (3900) `mosek.rescode.err_size_license_numcores`
The computer contains more cpu cores than the license allows for.
- (3910) `mosek.rescode.err_infeas_undefined`
The requested value is not defined for this solution type.
- (3999) `mosek.rescode.err_api_internal`
An internal fatal error occurred in an interface function.
- (4000) `mosek.rescode.trm_max_iterations`
The optimizer terminated at the maximum number of iterations.
- (4001) `mosek.rescode.trm_max_time`
The optimizer terminated at the maximum amount of time.
- (4002) `mosek.rescode.trm_objective_range`
The optimizer terminated on the bound of the objective range.
- (4003) `mosek.rescode.trm_mio_near_rel_gap`
The mixed-integer optimizer terminated because the near optimal relative gap tolerance was satisfied.
- (4004) `mosek.rescode.trm_mio_near_abs_gap`
The mixed-integer optimizer terminated because the near optimal absolute gap tolerance was satisfied.
- (4005) `mosek.rescode.trm_user_break`
Not in use.
- (4006) `mosek.rescode.trm_stall`
The optimizer terminated due to slow progress. The most likely reason causing slow progress is that the problem is badly formulated e.g. badly scaled or near infeasible. Sometimes a few dense columns in the constraint matrix can also lead to numerical problems that causes a stall.

The solution returned may or may not be of acceptable quality. Therefore, the solution status should be examined to determine the status of the solution. If the solution is near optimal, then for most practical purposes the solution will be good enough.

In particular, if a linear optimization problem is solved with the interior-point optimizer with basis identification turned on, the returned solution may be of acceptable quality, even in the optimizer stalled.
- (4007) `mosek.rescode.trm_user_callback`
The optimizer terminated due to the return of the user-defined call-back function.
- (4008) `mosek.rescode.trm_mio_num_relaxs`
The mixed-integer optimizer terminated as the maximum number of relaxations was reached.
- (4009) `mosek.rescode.trm_mio_num_branches`
The mixed-integer optimizer terminated as to the maximum number of branches was reached.

- (4015) `mosek.rescode.trm_num_max_num_int_solutions`
The mixed-integer optimizer terminated as the maximum number of feasible solutions was reached.
- (4020) `mosek.rescode.trm_max_num_setbacks`
The optimizer terminated as the maximum number of set-backs was reached. This indicates numerical problems and a possibly badly formulated problem.
- (4025) `mosek.rescode.trm_numerical_problem`
The optimizer terminated due to numerical problems.
- (4030) `mosek.rescode.trm_internal`
The optimizer terminated due to some internal reason. Please contact MOSEK support.
- (4031) `mosek.rescode.trm_internal_stop`
The optimizer terminated for internal reasons. Please contact MOSEK support.

Chapter 17

Constants

accmode	412
Constraint or variable access modes	
adopcode	412
Function opcode	
adoptype	413
Function operand type	
basindtype	413
Basis identification	
boundkey	413
Bound keys	
branchdir	414
Specifies the branching direction.	
callbackcode	414
Progress call-back codes	
checkconvexitytype	422
Types of convexity checks.	
compresstype	422
Compression types	
conetype	422
Cone types	
cputype	422
CPU type	
dataformat	423
Data format types	

dinfitem	424
Double information items	
feasrepairtype	428
Feasibility repair types	
feature	428
License feature	
iinfitem	428
Integer information items.	
inftype	435
Information item types	
iomode	435
Input/output modes	
language	435
Language selection constants	
liinfitem	435
Long integer information items.	
mark	436
Mark	
miocontsoltype	437
Continuous mixed-integer solution type	
miomode	437
Integer restrictions	
mionodeseltype	437
Mixed-integer node selection types	
mpsformat	438
MPS file format type	
msgkey	438
Message keys	
networkdetect	438
Network detection method	
objsense	438
Objective sense types	
onoffkey	439
On/off	
optimizertype	439
Optimizer types	

<code>orderingtype</code>	439
Ordering strategies	
<code>parametertype</code>	440
Parameter type	
<code>presolvemode</code>	440
Presolve method.	
<code>problemitem</code>	440
Problem data items	
<code>problemtypes</code>	441
Problem types	
<code>prosta</code>	441
Problem status keys	
<code>qreadtype</code>	442
Interpretation of quadratic terms in MPS files	
<code>rescodetype</code>	442
Response code type	
<code>scalingmethod</code>	443
Scaling type	
<code>scalingtype</code>	443
Scaling type	
<code>sensitivitytype</code>	443
Sensitivity types	
<code>simdegen</code>	443
Degeneracy strategies	
<code>simdupvec</code>	444
Exploit duplicate columns.	
<code>simhotstart</code>	444
Hot-start type employed by the simplex optimizer	
<code>simreform</code>	444
Problem reformulation.	
<code>simseltype</code>	444
Simplex selection strategy	
<code>solitem</code>	445
Solution items	
<code>solsta</code>	445
Solution status keys	

<code>soltype</code>	446
Solution types	
<code>solveform</code>	447
Solve primal or dual form	
<code>stakey</code>	447
Status keys	
<code>startpointtype</code>	447
Starting point types	
<code>streamtype</code>	448
Stream types	
<code>value</code>	448
Integer values	
<code>variabletype</code>	448
Variable types	
<code>xmlwriteroutputtype</code>	448
XML writer output mode	

17.1 Constraint or variable access modes

- (1) `mosek.accmode.con`
Access data by rows (constraint oriented)
- (0) `mosek.accmode.var`
Access data by columns (variable oriented)

17.2 Function opcode

- (0) `mosek.adopcode.add`
Add two operands.
- (3) `mosek.adopcode.div`
Divide two operands.
- (5) `mosek.adopcode.exp`
Exponential function of one operand.
- (6) `mosek.adopcode.log`
Logarithm function of one operand.
- (2) `mosek.adopcode.mul`
Multiply two operands.

- (4) `mosek.adopcode.pow`
First operand to the power the second operand.
- (7) `mosek.adopcode.ret`
Return one operand.
- (1) `mosek.adopcode.sub`
Subtract two operands.

17.3 Function operand type

- (1) `mosek.adoptype.constant`
Operand refers to a constant.
- (0) `mosek.adoptype.none`
Operand not used.
- (3) `mosek.adoptype.reference`
Operand refers to the result of another operation.
- (2) `mosek.adoptype.variable`
Operand refers to a variable.

17.4 Basis identification

- (1) `mosek.basindtype.always`
Basis identification is always performed even if the interior-point optimizer terminates abnormally.
- (3) `mosek.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.
- (0) `mosek.basindtype.never`
Never do basis identification.
- (2) `mosek.basindtype.no_error`
Basis identification is performed if the interior-point optimizer terminates without an error.
- (4) `mosek.basindtype.other`
Try another BI method.

17.5 Bound keys

- (3) `mosek.boundkey.fr`
The constraint or variable is free.

- (2) `mosek.boundkey.fx`
The constraint or variable is fixed.
- (0) `mosek.boundkey.lo`
The constraint or variable has a finite lower bound and an infinite upper bound.
- (4) `mosek.boundkey.ra`
The constraint or variable is ranged.
- (1) `mosek.boundkey.up`
The constraint or variable has an infinite lower bound and an finite upper bound.

17.6 Specifies the branching direction.

- (2) `mosek.branchdir.down`
The mixed-integer optimizer always chooses the down branch first.
- (0) `mosek.branchdir.free`
The mixed-integer optimizer decides which branch to choose.
- (1) `mosek.branchdir.up`
The mixed-integer optimizer always chooses the up branch first.

17.7 Progress call-back codes

- (0) `mosek.callbackcode.begin_bi`
The basis identification procedure has been started.
- (1) `mosek.callbackcode.begin_concurrent`
Concurrent optimizer is started.
- (2) `mosek.callbackcode.begin_conic`
The call-back function is called when the conic optimizer is started.
- (3) `mosek.callbackcode.begin_dual_bi`
The call-back function is called from within the basis identification procedure when the dual phase is started.
- (4) `mosek.callbackcode.begin_dual_sensitivity`
Dual sensitivity analysis is started.
- (5) `mosek.callbackcode.begin_dual_setup_bi`
The call-back function is called when the dual BI phase is started.
- (6) `mosek.callbackcode.begin_dual_simplex`
The call-back function is called when the dual simplex optimizer started.

- (7) `mosek.callbackcode.begin_dual_simplex_bi`
The call-back function is called from within the basis identification procedure when the dual simplex clean-up phase is started.
- (8) `mosek.callbackcode.begin_full_convexity_check`
Begin full convexity check.
- (9) `mosek.callbackcode.begin_infeas_ana`
The call-back function is called when the infeasibility analyzer is started.
- (10) `mosek.callbackcode.begin_intpnt`
The call-back function is called when the interior-point optimizer is started.
- (11) `mosek.callbackcode.begin_license_wait`
Begin waiting for license.
- (12) `mosek.callbackcode.begin_mio`
The call-back function is called when the mixed-integer optimizer is started.
- (13) `mosek.callbackcode.begin_network_dual_simplex`
The call-back function is called when the dual network simplex optimizer is started.
- (14) `mosek.callbackcode.begin_network_primal_simplex`
The call-back function is called when the primal network simplex optimizer is started.
- (15) `mosek.callbackcode.begin_network_simplex`
The call-back function is called when the simplex network optimizer is started.
- (16) `mosek.callbackcode.begin_nonconvex`
The call-back function is called when the nonconvex optimizer is started.
- (17) `mosek.callbackcode.begin_optimizer`
The call-back function is called when the optimizer is started.
- (18) `mosek.callbackcode.begin_presolve`
The call-back function is called when the presolve is started.
- (19) `mosek.callbackcode.begin_primal_bi`
The call-back function is called from within the basis identification procedure when the primal phase is started.
- (20) `mosek.callbackcode.begin_primal_dual_simplex`
The call-back function is called when the primal-dual simplex optimizer is started.
- (21) `mosek.callbackcode.begin_primal_dual_simplex_bi`
The call-back function is called from within the basis identification procedure when the primal-dual simplex clean-up phase is started.
- (22) `mosek.callbackcode.begin_primal_sensitivity`
Primal sensitivity analysis is started.

- (23) `mosek.callbackcode.begin_primal_setup_bi`
The call-back function is called when the primal BI setup is started.
- (24) `mosek.callbackcode.begin_primal_simplex`
The call-back function is called when the primal simplex optimizer is started.
- (25) `mosek.callbackcode.begin_primal_simplex_bi`
The call-back function is called from within the basis identification procedure when the primal simplex clean-up phase is started.
- (26) `mosek.callbackcode.begin_qcqp_reformulate`
Begin QCQP reformulation.
- (27) `mosek.callbackcode.begin_read`
MOSEK has started reading a problem file.
- (28) `mosek.callbackcode.begin_simplex`
The call-back function is called when the simplex optimizer is started.
- (29) `mosek.callbackcode.begin_simplex_bi`
The call-back function is called from within the basis identification procedure when the simplex clean-up phase is started.
- (30) `mosek.callbackcode.begin_simplex_network_detect`
The call-back function is called when the network detection procedure is started.
- (31) `mosek.callbackcode.begin_write`
MOSEK has started writing a problem file.
- (32) `mosek.callbackcode.conic`
The call-back function is called from within the conic optimizer after the information database has been updated.
- (33) `mosek.callbackcode.dual_simplex`
The call-back function is called from within the dual simplex optimizer.
- (34) `mosek.callbackcode.end_bi`
The call-back function is called when the basis identification procedure is terminated.
- (35) `mosek.callbackcode.end_concurrent`
Concurrent optimizer is terminated.
- (36) `mosek.callbackcode.end_conic`
The call-back function is called when the conic optimizer is terminated.
- (37) `mosek.callbackcode.end_dual_bi`
The call-back function is called from within the basis identification procedure when the dual phase is terminated.
- (38) `mosek.callbackcode.end_dual_sensitivity`
Dual sensitivity analysis is terminated.

- (39) `mosek.callbackcode.end_dual_setup_bi`
The call-back function is called when the dual BI phase is terminated.
- (40) `mosek.callbackcode.end_dual_simplex`
The call-back function is called when the dual simplex optimizer is terminated.
- (41) `mosek.callbackcode.end_dual_simplex_bi`
The call-back function is called from within the basis identification procedure when the dual clean-up phase is terminated.
- (42) `mosek.callbackcode.end_full_convexity_check`
End full convexity check.
- (43) `mosek.callbackcode.end_infeas_ana`
The call-back function is called when the infeasibility analyzer is terminated.
- (44) `mosek.callbackcode.end_intpnt`
The call-back function is called when the interior-point optimizer is terminated.
- (45) `mosek.callbackcode.end_license_wait`
End waiting for license.
- (46) `mosek.callbackcode.end_mio`
The call-back function is called when the mixed-integer optimizer is terminated.
- (47) `mosek.callbackcode.end_network_dual_simplex`
The call-back function is called when the dual network simplex optimizer is terminated.
- (48) `mosek.callbackcode.end_network_primal_simplex`
The call-back function is called when the primal network simplex optimizer is terminated.
- (49) `mosek.callbackcode.end_network_simplex`
The call-back function is called when the simplex network optimizer is terminated.
- (50) `mosek.callbackcode.end_nonconvex`
The call-back function is called when the nonconvex optimizer is terminated.
- (51) `mosek.callbackcode.end_optimizer`
The call-back function is called when the optimizer is terminated.
- (52) `mosek.callbackcode.end_presolve`
The call-back function is called when the presolve is completed.
- (53) `mosek.callbackcode.end_primal_bi`
The call-back function is called from within the basis identification procedure when the primal phase is terminated.
- (54) `mosek.callbackcode.end_primal_dual_simplex`
The call-back function is called when the primal-dual simplex optimizer is terminated.
- (55) `mosek.callbackcode.end_primal_dual_simplex_bi`
The call-back function is called from within the basis identification procedure when the primal-dual clean-up phase is terminated.

- (56) `mosek.callbackcode.end_primal_sensitivity`
Primal sensitivity analysis is terminated.
- (57) `mosek.callbackcode.end_primal_setup_bi`
The call-back function is called when the primal BI setup is terminated.
- (58) `mosek.callbackcode.end_primal_simplex`
The call-back function is called when the primal simplex optimizer is terminated.
- (59) `mosek.callbackcode.end_primal_simplex_bi`
The call-back function is called from within the basis identification procedure when the primal clean-up phase is terminated.
- (60) `mosek.callbackcode.end_qcqp_reformulate`
End QCQP reformulation.
- (61) `mosek.callbackcode.end_read`
MOSEK has finished reading a problem file.
- (62) `mosek.callbackcode.end_simplex`
The call-back function is called when the simplex optimizer is terminated.
- (63) `mosek.callbackcode.end_simplex_bi`
The call-back function is called from within the basis identification procedure when the simplex clean-up phase is terminated.
- (64) `mosek.callbackcode.end_simplex_network_detect`
The call-back function is called when the network detection procedure is terminated.
- (65) `mosek.callbackcode.end_write`
MOSEK has finished writing a problem file.
- (66) `mosek.callbackcode.im_bi`
The call-back function is called from within the basis identification procedure at an intermediate point.
- (67) `mosek.callbackcode.im_conic`
The call-back function is called at an intermediate stage within the conic optimizer where the information database has not been updated.
- (68) `mosek.callbackcode.im_dual_bi`
The call-back function is called from within the basis identification procedure at an intermediate point in the dual phase.
- (69) `mosek.callbackcode.im_dual_sensitivity`
The call-back function is called at an intermediate stage of the dual sensitivity analysis.
- (70) `mosek.callbackcode.im_dual_simplex`
The call-back function is called at an intermediate point in the dual simplex optimizer.
- (71) `mosek.callbackcode.im_full_convexity_check`
The call-back function is called at an intermediate stage of the full convexity check.

- (72) `mosek.callbackcode.im_intpnt`
The call-back function is called at an intermediate stage within the interior-point optimizer where the information database has not been updated.
- (73) `mosek.callbackcode.im_license_wait`
MOSEK is waiting for a license.
- (74) `mosek.callbackcode.im_lu`
The call-back function is called from within the LU factorization procedure at an intermediate point.
- (75) `mosek.callbackcode.im_mio`
The call-back function is called at an intermediate point in the mixed-integer optimizer.
- (76) `mosek.callbackcode.im_mio_dual_simplex`
The call-back function is called at an intermediate point in the mixed-integer optimizer while running the dual simplex optimizer.
- (77) `mosek.callbackcode.im_mio_intpnt`
The call-back function is called at an intermediate point in the mixed-integer optimizer while running the interior-point optimizer.
- (78) `mosek.callbackcode.im_mio_presolve`
The call-back function is called at an intermediate point in the mixed-integer optimizer while running the presolve.
- (79) `mosek.callbackcode.im_mio_primal_simplex`
The call-back function is called at an intermediate point in the mixed-integer optimizer while running the primal simplex optimizer.
- (80) `mosek.callbackcode.im_network_dual_simplex`
The call-back function is called at an intermediate point in the dual network simplex optimizer.
- (81) `mosek.callbackcode.im_network_primal_simplex`
The call-back function is called at an intermediate point in the primal network simplex optimizer.
- (82) `mosek.callbackcode.im_nonconvex`
The call-back function is called at an intermediate stage within the nonconvex optimizer where the information database has not been updated.
- (83) `mosek.callbackcode.im_order`
The call-back function is called from within the matrix ordering procedure at an intermediate point.
- (84) `mosek.callbackcode.im_presolve`
The call-back function is called from within the presolve procedure at an intermediate stage.
- (85) `mosek.callbackcode.im_primal_bi`
The call-back function is called from within the basis identification procedure at an intermediate point in the primal phase.

- (86) `mosek.callbackcode.im_primal_dual_simplex`
The call-back function is called at an intermediate point in the primal-dual simplex optimizer.
- (87) `mosek.callbackcode.im_primal_sensitivity`
The call-back function is called at an intermediate stage of the primal sensitivity analysis.
- (88) `mosek.callbackcode.im_primal_simplex`
The call-back function is called at an intermediate point in the primal simplex optimizer.
- (89) `mosek.callbackcode.im_qo_reformulate`
The call-back function is called at an intermediate stage of the QP to SOCP reformulation.
- (90) `mosek.callbackcode.im_simplex`
The call-back function is called from within the simplex optimizer at an intermediate point.
- (91) `mosek.callbackcode.im_simplex_bi`
The call-back function is called from within the basis identification procedure at an intermediate point in the simplex clean-up phase. The frequency of the call-backs is controlled by the `mosek.iparam.log_sim_freq` parameter.
- (92) `mosek.callbackcode.intpnt`
The call-back function is called from within the interior-point optimizer after the information database has been updated.
- (93) `mosek.callbackcode.new_int_mio`
The call-back function is called after a new integer solution has been located by the mixed-integer optimizer.
- (94) `mosek.callbackcode.nonconvex`
The call-back function is called from within the nonconvex optimizer after the information database has been updated.
- (95) `mosek.callbackcode.primal_simplex`
The call-back function is called from within the primal simplex optimizer.
- (96) `mosek.callbackcode.qcone`
The call-back function is called from within the Qcone optimizer.
- (97) `mosek.callbackcode.read_add_anz`
A chunk of A non-zeos has been read from a problem file.
- (98) `mosek.callbackcode.read_add_cones`
A chunk of cones has been read from a problem file.
- (99) `mosek.callbackcode.read_add_cons`
A chunk of constraints has been read from a problem file.
- (100) `mosek.callbackcode.read_add_qnz`
A chunk of Q non-zeos has been read from a problem file.
- (101) `mosek.callbackcode.read_add_vars`
A chunk of variables has been read from a problem file.

- (102) `mosek.callbackcode.read_opf`
The call-back function is called from the OPF reader.
- (103) `mosek.callbackcode.read_opf_section`
A chunk of Q non-zeos has been read from a problem file.
- (104) `mosek.callbackcode.update_dual_bi`
The call-back function is called from within the basis identification procedure at an intermediate point in the dual phase.
- (105) `mosek.callbackcode.update_dual_simplex`
The call-back function is called in the dual simplex optimizer.
- (106) `mosek.callbackcode.update_dual_simplex.bi`
The call-back function is called from within the basis identification procedure at an intermediate point in the dual simplex clean-up phase. The frequency of the call-backs is controlled by the `mosek.iparam.log_sim_freq` parameter.
- (107) `mosek.callbackcode.update_network_dual_simplex`
The call-back function is called in the dual network simplex optimizer.
- (108) `mosek.callbackcode.update_network_primal_simplex`
The call-back function is called in the primal network simplex optimizer.
- (109) `mosek.callbackcode.update_nonconvex`
The call-back function is called at an intermediate stage within the nonconvex optimizer where the information database has been updated.
- (110) `mosek.callbackcode.update_presolve`
The call-back function is called from within the presolve procedure.
- (111) `mosek.callbackcode.update_primal_bi`
The call-back function is called from within the basis identification procedure at an intermediate point in the primal phase.
- (112) `mosek.callbackcode.update_primal_dual_simplex`
The call-back function is called in the primal-dual simplex optimizer.
- (113) `mosek.callbackcode.update_primal_dual_simplex.bi`
The call-back function is called from within the basis identification procedure at an intermediate point in the primal-dual simplex clean-up phase. The frequency of the call-backs is controlled by the `mosek.iparam.log_sim_freq` parameter.
- (114) `mosek.callbackcode.update_primal_simplex`
The call-back function is called in the primal simplex optimizer.
- (115) `mosek.callbackcode.update_primal_simplex.bi`
The call-back function is called from within the basis identification procedure at an intermediate point in the primal simplex clean-up phase. The frequency of the call-backs is controlled by the `mosek.iparam.log_sim_freq` parameter.
- (116) `mosek.callbackcode.write_opf`
The call-back function is called from the OPF writer.

17.8 Types of convexity checks.

- (2) `mosek.checkconvexitytype.full`
Perform a full convexity check.
- (0) `mosek.checkconvexitytype.none`
No convexity check.
- (1) `mosek.checkconvexitytype.simple`
Perform simple and fast convexity check.

17.9 Compression types

- (1) `mosek.compresstype.free`
The type of compression used is chosen automatically.
- (2) `mosek.compresstype.gzip`
The type of compression used is gzip compatible.
- (0) `mosek.compresstype.none`
No compression is used.

17.10 Cone types

- (0) `mosek.conetype.quad`
The cone is a quadratic cone.
- (1) `mosek.conetype.rquad`
The cone is a rotated quadratic cone.

17.11 CPU type

- (4) `mosek.cputype.amd_athlon`
An AMD Athlon.
- (7) `mosek.cputype.amd_opteron`
An AMD Opteron (64 bit).
- (1) `mosek.cputype.generic`
An generic CPU type for the platform
- (5) `mosek.cputype.hp_parisc20`
An HP PA RISC version 2.0 CPU.

- (10) `mosek.cputype.intel_core2`
An Intel CORE2 cpu.
- (6) `mosek.cputype.intel_itanium2`
An Intel Itanium2.
- (2) `mosek.cputype.intel_p3`
An Intel Pentium P3.
- (3) `mosek.cputype.intel_p4`
An Intel Pentium P4 or Intel Xeon.
- (9) `mosek.cputype.intel_pm`
An Intel PM cpu.
- (8) `mosek.cputype.powerpc_g5`
A G5 PowerPC CPU.
- (0) `mosek.cputype.unknown`
An unknown CPU.

17.12 Data format types

- (0) `mosek.dataformat.extension`
The file extension is used to determine the data file format.
- (6) `mosek.dataformat.free_mps`
The data data a free MPS formatted file.
- (2) `mosek.dataformat.lp`
The data file is LP formatted.
- (3) `mosek.dataformat.mbt`
The data file is a MOSEK binary task file.
- (1) `mosek.dataformat.mps`
The data file is MPS formatted.
- (4) `mosek.dataformat.op`
The data file is an optimization problem formatted file.
- (5) `mosek.dataformat.xml`
The data file is an XML formatted file.

17.13 Double information items

- (0) `mosek.dinfitem.bi_clean_dual_time`
Time spent within the dual clean-up optimizer of the basis identification procedure since its invocation.
- (1) `mosek.dinfitem.bi_clean_primal_dual_time`
Time spent within the primal-dual clean-up optimizer of the basis identification procedure since its invocation.
- (2) `mosek.dinfitem.bi_clean_primal_time`
Time spent within the primal clean-up optimizer of the basis identification procedure since its invocation.
- (3) `mosek.dinfitem.bi_clean_time`
Time spent within the clean-up phase of the basis identification procedure since its invocation.
- (4) `mosek.dinfitem.bi_dual_time`
Time spent within the dual phase basis identification procedure since its invocation.
- (5) `mosek.dinfitem.bi_primal_time`
Time spent within the primal phase of the basis identification procedure since its invocation.
- (6) `mosek.dinfitem.bi_time`
Time spent within the basis identification procedure since its invocation.
- (7) `mosek.dinfitem.concurrent_time`
Time spent within the concurrent optimizer since its invocation.
- (8) `mosek.dinfitem.intpnt_dual_feas`
Dual feasibility measure reported by the interior-point and Qcone optimizer. (For the interior-point optimizer this measure does not directly related to the original problem because a homogeneous model is employed.)
- (9) `mosek.dinfitem.intpnt_dual_obj`
Dual objective value reported by the interior-point or Qcone optimizer.
- (10) `mosek.dinfitem.intpnt_factor_num_flops`
An estimate of the number of flops used in the factorization.
- (11) `mosek.dinfitem.intpnt_kap_div_tau`
This measure should converge to zero if the problem has a primal-dual optimal solution or to infinity if problem is (strictly) primal or dual infeasible. In case the measure is converging towards a positive but bounded constant the problem is usually ill-posed.
- (12) `mosek.dinfitem.intpnt_order_time`
Order time (in seconds).
- (13) `mosek.dinfitem.intpnt_primal_feas`
Primal feasibility measure reported by the interior-point or Qcone optimizers. (For the interior-point optimizer this measure does not directly related to the original problem because a homogeneous model is employed).

- (14) `mosek.dinfitem.intpnt_primal_obj`
Primal objective value reported by the interior-point or Qcone optimizer.
- (15) `mosek.dinfitem.intpnt_time`
Time spent within the interior-point optimizer since its invocation.
- (16) `mosek.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.
- (17) `mosek.dinfitem.mio_heuristic_time`
Time spent in the optimizer while solving the relaxations.
- (18) `mosek.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.

- (19) `mosek.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 `mosek.iinfitem.mio_num_relax` is strictly positive.
- (20) `mosek.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 located i.e. check `mosek.iinfitem.mio_num_int_solutions`.
- (21) `mosek.dinfitem.mio_obj_rel_gap`
Given that the mixed-integer optimizer has computed a feasible solution and a bound on the optimal objective value, then this item contains the relative gap defined by

$$\frac{|(\text{objective value of feasible solution}) - (\text{objective bound})|}{\max(\delta, |(\text{objective value of feasible solution})|)}.$$

where δ is given by the parameter `mosek.dparam.mio_rel_gap_const`. Otherwise it has the value -1.0.

- (22) `mosek.dinfitem.mio_optimizer_time`
Time spent in the optimizer while solving the relaxations.
- (23) `mosek.dinfitem.mio_root_optimizer_time`
Time spent in the optimizer while solving the root relaxation.
- (24) `mosek.dinfitem.mio_root_presolve_time`
Time spent in while presolveing the root relaxation.
- (25) `mosek.dinfitem.mio_time`
Time spent in the mixed-integer optimizer.

- (26) `mosek.dinfitem.mio_user_obj_cut`
If the objective cut is used, then this information item has the value of the cut.
- (27) `mosek.dinfitem.optimizer_time`
Total time spent in the optimizer since it was invoked.
- (28) `mosek.dinfitem.presolve_eli_time`
Total time spent in the eliminator since the presolve was invoked.
- (29) `mosek.dinfitem.presolve_lindep_time`
Total time spent in the linear dependency checker since the presolve was invoked.
- (30) `mosek.dinfitem.presolve_time`
Total time (in seconds) spent in the presolve since it was invoked.
- (31) `mosek.dinfitem.qcqp_reformulate_time`
Time spent with QP reformulation.
- (32) `mosek.dinfitem.rd_time`
Time spent reading the data file.
- (33) `mosek.dinfitem.sim_dual_time`
Time spent in the dual simplex optimizer since invoking it.
- (34) `mosek.dinfitem.sim_feas`
Feasibility measure reported by the simplex optimizer.
- (35) `mosek.dinfitem.sim_network_dual_time`
Time spent in the dual network simplex optimizer since invoking it.
- (36) `mosek.dinfitem.sim_network_primal_time`
Time spent in the primal network simplex optimizer since invoking it.
- (37) `mosek.dinfitem.sim_network_time`
Time spent in the network simplex optimizer since invoking it.
- (38) `mosek.dinfitem.sim_obj`
Objective value reported by the simplex optimizer.
- (39) `mosek.dinfitem.sim_primal_dual_time`
Time spent in the primal-dual simplex optimizer since invoking it.
- (40) `mosek.dinfitem.sim_primal_time`
Time spent in the primal simplex optimizer since invoking it.
- (41) `mosek.dinfitem.sim_time`
Time spent in the simplex optimizer since invoking it.
- (42) `mosek.dinfitem.sol_bas_dual_obj`
Dual objective value of the basic solution. Updated at the end of the optimization.
- (43) `mosek.dinfitem.sol_bas_max_dbi`
Maximal dual bound infeasibility in the basic solution. Updated at the end of the optimization.

- (44) `mosek.dinfitem.sol_bas_max_deqi`
Maximal dual equality infeasibility in the basic solution. Updated at the end of the optimization.
- (45) `mosek.dinfitem.sol_bas_max_pbi`
Maximal primal bound infeasibility in the basic solution. Updated at the end of the optimization.
- (46) `mosek.dinfitem.sol_bas_max_peqi`
Maximal primal equality infeasibility in the basic solution. Updated at the end of the optimization.
- (47) `mosek.dinfitem.sol_bas_max_pinti`
Maximal primal integer infeasibility in the basic solution. Updated at the end of the optimization.
- (48) `mosek.dinfitem.sol_bas_primal_obj`
Primal objective value of the basic solution. Updated at the end of the optimization.
- (49) `mosek.dinfitem.sol_int_max_pbi`
Maximal primal bound infeasibility in the integer solution. Updated at the end of the optimization.
- (50) `mosek.dinfitem.sol_int_max_peqi`
Maximal primal equality infeasibility in the basic solution. Updated at the end of the optimization.
- (51) `mosek.dinfitem.sol_int_max_pinti`
Maximal primal integer infeasibility in the integer solution. Updated at the end of the optimization.
- (52) `mosek.dinfitem.sol_int_primal_obj`
Primal objective value of the integer solution. Updated at the end of the optimization.
- (53) `mosek.dinfitem.sol_itr_dual_obj`
Dual objective value of the interior-point solution. Updated at the end of the optimization.
- (54) `mosek.dinfitem.sol_itr_max_dbi`
Maximal dual bound infeasibility in the interior-point solution. Updated at the end of the optimization.
- (55) `mosek.dinfitem.sol_itr_max_dcni`
Maximal dual cone infeasibility in the interior-point solution. Updated at the end of the optimization.
- (56) `mosek.dinfitem.sol_itr_max_deqi`
Maximal dual equality infeasibility in the interior-point solution. Updated at the end of the optimization.
- (57) `mosek.dinfitem.sol_itr_max_pbi`
Maximal primal bound infeasibility in the interior-point solution. Updated at the end of the optimization.

- (58) `mosek.dinfitem.sol_itr_max_pcni`
Maximal primal cone infeasibility in the interior-point solution. Updated at the end of the optimization.
- (59) `mosek.dinfitem.sol_itr_max_peqi`
Maximal primal equality infeasibility in the interior-point solution. Updated at the end of the optimization.
- (60) `mosek.dinfitem.sol_itr_max_pinti`
Maximal primal integer infeasibility in the interior-point solution. Updated at the end of the optimization.
- (61) `mosek.dinfitem.sol_itr_primal_obj`
Primal objective value of the interior-point solution. Updated at the end of the optimization.

17.14 Feasibility repair types

- (2) `mosek.feasrepairtype.optimize_combined`
Minimize with original objective subject to minimal weighted violation of bounds.
- (0) `mosek.feasrepairtype.optimize_none`
Do not optimize the feasibility repair problem.
- (1) `mosek.feasrepairtype.optimize_penalty`
Minimize weighted sum of violations.

17.15 License feature

- (2) `mosek.feature.ptom`
Mixed-integer extension.
- (1) `mosek.feature.pton`
Nonlinear extension.
- (3) `mosek.feature.ptox`
Non-convex extension.
- (0) `mosek.feature.pts`
Base system.

17.16 Integer information items.

- (0) `mosek.iinfitem.ana_pro_num_con`
Number of constraints in the problem.
This value is set by `Task.analyzeproblem`.

- (1) `mosek.iinfitem.ana_pro_num_con_eq`
Number of equality constraints.
This value is set by `Task.analyzeproblem`.
- (2) `mosek.iinfitem.ana_pro_num_con_fr`
Number of unbounded constraints.
This value is set by `Task.analyzeproblem`.
- (3) `mosek.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`.
- (4) `mosek.iinfitem.ana_pro_num_con_ra`
Number of constraints with finite lower and upper bounds.
This value is set by `Task.analyzeproblem`.
- (5) `mosek.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`.
- (6) `mosek.iinfitem.ana_pro_num_var`
Number of variables in the problem.
This value is set by `Task.analyzeproblem`.
- (7) `mosek.iinfitem.ana_pro_num_var_bin`
Number of binary (0-1) variables.
This value is set by `Task.analyzeproblem`.
- (8) `mosek.iinfitem.ana_pro_num_var_cont`
Number of continuous variables.
This value is set by `Task.analyzeproblem`.
- (9) `mosek.iinfitem.ana_pro_num_var_eq`
Number of fixed variables.
This value is set by `Task.analyzeproblem`.
- (10) `mosek.iinfitem.ana_pro_num_var_fr`
Number of free variables.
This value is set by `Task.analyzeproblem`.
- (11) `mosek.iinfitem.ana_pro_num_var_int`
Number of general integer variables.
This value is set by `Task.analyzeproblem`.
- (12) `mosek.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`.

- (13) `mosek.iinfitem.ana_pro_num_var_ra`
Number of variables with finite lower and upper bounds.
This value is set by `Task.analyzeproblem`.
- (14) `mosek.iinfitem.ana_pro_num_var_up`
Number of variables with an upper bound and an infinite lower bound. This value is set by
This value is set by `Task.analyzeproblem`.
- (15) `mosek.iinfitem.cache_size_l1`
L1 cache size used.
- (16) `mosek.iinfitem.cache_size_l2`
L2 cache size used.
- (17) `mosek.iinfitem.concurrent_fastest_optimizer`
The type of the optimizer that finished first in a concurrent optimization.
- (18) `mosek.iinfitem.cpu_type`
The type of cpu detected.
- (19) `mosek.iinfitem.intpnt_factor_num_offcol`
Number of columns in the constraint matrix (or Jacobian) that has an offending structure.
- (20) `mosek.iinfitem.intpnt_iter`
Number of interior-point iterations since invoking the interior-point optimizer.
- (21) `mosek.iinfitem.intpnt_num_threads`
Number of threads that the interior-point optimizer is using.
- (22) `mosek.iinfitem.intpnt_solve_dual`
Non-zero if the interior-point optimizer is solving the dual problem.
- (23) `mosek.iinfitem.mio_construct_solution`
If this item has the value 0, then MOSEK did not try to construct an initial integer feasible solution. If the item has a positive value, then MOSEK successfully constructed an initial integer feasible solution.
- (24) `mosek.iinfitem.mio_initial_solution`
Is non-zero if an initial integer solution is specified.
- (25) `mosek.iinfitem.mio_num_active_nodes`
Number of active nodes in the branch and bound tree.
- (26) `mosek.iinfitem.mio_num_branch`
Number of branches performed during the optimization.
- (27) `mosek.iinfitem.mio_num_cuts`
Number of cuts generated by the mixed-integer optimizer.
- (28) `mosek.iinfitem.mio_num_int_solutions`
Number of integer feasible solutions that has been found.

- (29) `mosek.iinfitem.mio_num_relax`
Number of relaxations solved during the optimization.
- (30) `mosek.iinfitem.mio_numcon`
Number of constraints in the problem solved by the mixed-integer optimizer.
- (31) `mosek.iinfitem.mio_numint`
Number of integer variables in the problem solved by the mixed-integer optimizer.
- (32) `mosek.iinfitem.mio_numvar`
Number of variables in the problem solved by the mixed-integer optimizer.
- (33) `mosek.iinfitem.mio_total_num_basis_cuts`
Number of basis cuts.
- (34) `mosek.iinfitem.mio_total_num_branch`
Number of branches performed during the optimization.
- (35) `mosek.iinfitem.mio_total_num_cardgub_cuts`
Number of cardgub cuts.
- (36) `mosek.iinfitem.mio_total_num_clique_cuts`
Number of clique cuts.
- (37) `mosek.iinfitem.mio_total_num_coef_redc_cuts`
Number of coef. redc. cuts.
- (38) `mosek.iinfitem.mio_total_num_contra_cuts`
Number of contra cuts.
- (39) `mosek.iinfitem.mio_total_num_cuts`
Total number of cuts generated by the mixed-integer optimizer.
- (40) `mosek.iinfitem.mio_total_num_disagg_cuts`
Number of diasagg cuts.
- (41) `mosek.iinfitem.mio_total_num_flow_cover_cuts`
Number of flow cover cuts.
- (42) `mosek.iinfitem.mio_total_num_gcd_cuts`
Number of gcd cuts.
- (43) `mosek.iinfitem.mio_total_num_gomory_cuts`
Number of Gomory cuts.
- (44) `mosek.iinfitem.mio_total_num_gub_cover_cuts`
Number of GUB cover cuts.
- (45) `mosek.iinfitem.mio_total_num_knapsur_cover_cuts`
Number of knapsack cover cuts.
- (46) `mosek.iinfitem.mio_total_num_lattice_cuts`
Number of lattice cuts.

- (47) `mosek.iinfitem.mio_total_num_lift_cuts`
Number of lift cuts.
- (48) `mosek.iinfitem.mio_total_num_obj_cuts`
Number of obj cuts.
- (49) `mosek.iinfitem.mio_total_num_plan_loc_cuts`
Number of loc cuts.
- (50) `mosek.iinfitem.mio_total_num_relax`
Number of relaxations solved during the optimization.
- (51) `mosek.iinfitem.mio_user_obj_cut`
If it is non-zero, then the objective cut is used.
- (52) `mosek.iinfitem.opt_numcon`
Number of constraints in the problem solved when the optimizer is called.
- (53) `mosek.iinfitem.opt_numvar`
Number of variables in the problem solved when the optimizer is called
- (54) `mosek.iinfitem.optimize_response`
The reponse code returned by optimize.
- (55) `mosek.iinfitem.rd_numcon`
Number of constraints read.
- (56) `mosek.iinfitem.rd_numcone`
Number of conic constraints read.
- (57) `mosek.iinfitem.rd_numintvar`
Number of integer-constrained variables read.
- (58) `mosek.iinfitem.rd_numq`
Number of nonempty Q matrices read.
- (59) `mosek.iinfitem.rd_numvar`
Number of variables read.
- (60) `mosek.iinfitem.rd_prototype`
Problem type.
- (61) `mosek.iinfitem.sim_dual_deg_iter`
The number of dual degenerate iterations.
- (62) `mosek.iinfitem.sim_dual_hotstart`
If 1 then the dual simplex algorithm is solving from an advanced basis.
- (63) `mosek.iinfitem.sim_dual_hotstart_lu`
If 1 then a valid basis factorization of full rank was located and used by the dual simplex algorithm.

- (64) `mosek.iinfitem.sim_dual_inf_iter`
The number of iterations taken with dual infeasibility.
- (65) `mosek.iinfitem.sim_dual_iter`
Number of dual simplex iterations during the last optimization.
- (66) `mosek.iinfitem.sim_network_dual_deg_iter`
The number of dual network degenerate iterations.
- (67) `mosek.iinfitem.sim_network_dual_hotstart`
If 1 then the dual network simplex algorithm is solving from an advanced basis.
- (68) `mosek.iinfitem.sim_network_dual_hotstart_lu`
If 1 then a valid basis factorization of full rank was located and used by the dual network simplex algorithm.
- (69) `mosek.iinfitem.sim_network_dual_inf_iter`
The number of iterations taken with dual infeasibility in the network optimizer.
- (70) `mosek.iinfitem.sim_network_dual_iter`
Number of dual network simplex iterations during the last optimization.
- (71) `mosek.iinfitem.sim_network_primal_deg_iter`
The number of primal network degenerate iterations.
- (72) `mosek.iinfitem.sim_network_primal_hotstart`
If 1 then the primal network simplex algorithm is solving from an advanced basis.
- (73) `mosek.iinfitem.sim_network_primal_hotstart_lu`
If 1 then a valid basis factorization of full rank was located and used by the primal network simplex algorithm.
- (74) `mosek.iinfitem.sim_network_primal_inf_iter`
The number of iterations taken with primal infeasibility in the network optimizer.
- (75) `mosek.iinfitem.sim_network_primal_iter`
Number of primal network simplex iterations during the last optimization.
- (76) `mosek.iinfitem.sim_numcon`
Number of constraints in the problem solved by the simplex optimizer.
- (77) `mosek.iinfitem.sim_numvar`
Number of variables in the problem solved by the simplex optimizer.
- (78) `mosek.iinfitem.sim_primal_deg_iter`
The number of primal degenerate iterations.
- (79) `mosek.iinfitem.sim_primal_dual_deg_iter`
The number of degenerate major iterations taken by the primal dual simplex algorithm.
- (80) `mosek.iinfitem.sim_primal_dual_hotstart`
If 1 then the primal dual simplex algorithm is solving from an advanced basis.

- (81) `mosek.iinfitem.sim_primal_dual_hotstart_lu`
If 1 then a valid basis factorization of full rank was located and used by the primal dual simplex algorithm.
- (82) `mosek.iinfitem.sim_primal_dual_inf_iter`
The number of master iterations with dual infeasibility taken by the primal dual simplex algorithm.
- (83) `mosek.iinfitem.sim_primal_dual_iter`
Number of primal dual simplex iterations during the last optimization.
- (84) `mosek.iinfitem.sim_primal_hotstart`
If 1 then the primal simplex algorithm is solving from an advanced basis.
- (85) `mosek.iinfitem.sim_primal_hotstart_lu`
If 1 then a valid basis factorization of full rank was located and used by the primal simplex algorithm.
- (86) `mosek.iinfitem.sim_primal_inf_iter`
The number of iterations taken with primal infeasibility.
- (87) `mosek.iinfitem.sim_primal_iter`
Number of primal simplex iterations during the last optimization.
- (88) `mosek.iinfitem.sim_solve_dual`
Is non-zero if dual problem is solved.
- (89) `mosek.iinfitem.sol_bas_prosta`
Problem status of the basic solution. Updated after each optimization.
- (90) `mosek.iinfitem.sol_bas_solsta`
Solution status of the basic solution. Updated after each optimization.
- (91) `mosek.iinfitem.sol_int_prosta`
Problem status of the integer solution. Updated after each optimization.
- (92) `mosek.iinfitem.sol_int_solsta`
Solution status of the integer solution. Updated after each optimization.
- (93) `mosek.iinfitem.sol_itr_prosta`
Problem status of the interior-point solution. Updated after each optimization.
- (94) `mosek.iinfitem.sol_itr_solsta`
Solution status of the interior-point solution. Updated after each optimization.
- (95) `mosek.iinfitem.sto_num_a_cache_flushes`
Number of times the cache of A elements is flushed. A large number implies that `maxnumanz` is too small as well as an inefficient usage of MOSEK.
- (96) `mosek.iinfitem.sto_num_a_realloc`
Number of times the storage for storing A has been changed. A large value may indicates that memory fragmentation may occur.

- (97) `mosek.iinfitem.sto_num_a_transposes`
Number of times the A matrix is transposed. A large number implies that `maxnumanz` is too small or an inefficient usage of MOSEK. This will occur in particular if the code alternate between accessing rows and columns of A .

17.17 Information item types

- (0) `mosek.inftype.dou_type`
Is a double information type.
- (1) `mosek.inftype.int_type`
Is an integer.
- (2) `mosek.inftype.lint_type`
Is a long integer.

17.18 Input/output modes

- (0) `mosek.iomode.read`
The file is read-only.
- (2) `mosek.iomode.readwrite`
The file is to read and written.
- (1) `mosek.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.

17.19 Language selection constants

- (1) `mosek.language.dan`
Danish language selection
- (0) `mosek.language.eng`
English language selection

17.20 Long integer information items.

- (0) `mosek.liinfitem.bi_clean_dual_deg_iter`
Number of dual degenerate clean iterations performed in the basis identification.
- (1) `mosek.liinfitem.bi_clean_dual_iter`
Number of dual clean iterations performed in the basis identification.

- (2) `mosek.liinfitem.bi_clean_primal_deg_iter`
Number of primal degenerate clean iterations performed in the basis identification.
- (3) `mosek.liinfitem.bi_clean_primal_dual_deg_iter`
Number of primal-dual degenerate clean iterations performed in the basis identification.
- (4) `mosek.liinfitem.bi_clean_primal_dual_iter`
Number of primal-dual clean iterations performed in the basis identification.
- (5) `mosek.liinfitem.bi_clean_primal_dual_sub_iter`
Number of primal-dual subproblem clean iterations performed in the basis identification.
- (6) `mosek.liinfitem.bi_clean_primal_iter`
Number of primal clean iterations performed in the basis identification.
- (7) `mosek.liinfitem.bi_dual_iter`
Number of dual pivots performed in the basis identification.
- (8) `mosek.liinfitem.bi_primal_iter`
Number of primal pivots performed in the basis identification.
- (9) `mosek.liinfitem.intpnt_factor_num_nz`
Number of non-zeros in factorization.
- (10) `mosek.liinfitem.mio_intpnt_iter`
Number of interior-point iterations performed by the mixed-integer optimizer.
- (11) `mosek.liinfitem.mio_simplex_iter`
Number of simplex iterations performed by the mixed-integer optimizer.
- (12) `mosek.liinfitem.rd_numanz`
Number of non-zeros in A that is read.
- (13) `mosek.liinfitem.rd_numqnz`
Number of Q non-zeros.

17.21 Mark

- (0) `mosek.mark.lo`
The lower bound is selected for sensitivity analysis.
- (1) `mosek.mark.up`
The upper bound is selected for sensitivity analysis.

17.22 Continuous mixed-integer solution type

(2) `mosek.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.

(3) `mosek.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.

(0) `mosek.miocontsoltype.none`

No interior-point or basic solution are reported when the mixed-integer optimizer is used.

(1) `mosek.miocontsoltype.root`

The reported interior-point and basic solutions are a solution to the root node problem when mixed-integer optimizer is used.

17.23 Integer restrictions

(0) `mosek.miomode.ignored`

The integer constraints are ignored and the problem is solved as a continuous problem.

(2) `mosek.miomode.lazy`

Integer restrictions should be satisfied if an optimizer is available for the problem.

(1) `mosek.miomode.satisfied`

Integer restrictions should be satisfied.

17.24 Mixed-integer node selection types

(2) `mosek.mionodeseltype.best`

The optimizer employs a best bound node selection strategy.

(1) `mosek.mionodeseltype.first`

The optimizer employs a depth first node selection strategy.

(0) `mosek.mionodeseltype.free`

The optimizer decides the node selection strategy.

(4) `mosek.mionodeseltype.hybrid`

The optimizer employs a hybrid strategy.

(5) `mosek.mionodeseltype.pseudo`

The optimizer employs selects the node based on a pseudo cost estimate.

- (3) `mosek.mionodeseltype.worst`
The optimizer employs a worst bound node selection strategy.

17.25 MPS file format type

- (2) `mosek.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.
- (1) `mosek.mpsformat.relaxed`
It is assumed that the input file satisfies a slightly relaxed version of the MPS format.
- (0) `mosek.mpsformat.strict`
It is assumed that the input file satisfies the MPS format strictly.

17.26 Message keys

- (1100) `mosek.msgkey.mps_selected`
- (1000) `mosek.msgkey.reading_file`
- (1001) `mosek.msgkey.writing_file`

17.27 Network detection method

- (2) `mosek.networkdetect.advanced`
The network detection should use a more advanced heuristic.
- (0) `mosek.networkdetect.free`
The network detection is free.
- (1) `mosek.networkdetect.simple`
The network detection should use a very simple heuristic.

17.28 Objective sense types

- (2) `mosek.objsense.maximize`
The problem should be maximized.
- (1) `mosek.objsense.minimize`
The problem should be minimized.
- (0) `mosek.objsense.undefined`
The objective sense is undefined.

17.29 On/off

- (0) `mosek.onoffkey.off`
Switch the option off.
- (1) `mosek.onoffkey.on`
Switch the option on.

17.30 Optimizer types

- (10) `mosek.optimizertype.concurrent`
The optimizer for nonconvex nonlinear problems.
- (2) `mosek.optimizertype.conic`
The optimizer for problems having conic constraints.
- (5) `mosek.optimizertype.dual_simplex`
The dual simplex optimizer is used.
- (0) `mosek.optimizertype.free`
The optimizer is chosen automatically.
- (7) `mosek.optimizertype.free_simplex`
One of the simplex optimizers is used.
- (1) `mosek.optimizertype.intpnt`
The interior-point optimizer is used.
- (8) `mosek.optimizertype.mixed_int`
The mixed-integer optimizer.
- (9) `mosek.optimizertype.nonconvex`
The optimizer for nonconvex nonlinear problems.
- (6) `mosek.optimizertype.primal_dual_simplex`
The primal dual simplex optimizer is used.
- (4) `mosek.optimizertype.primal_simplex`
The primal simplex optimizer is used.
- (3) `mosek.optimizertype.qcone`
For internal use only.

17.31 Ordering strategies

- (1) `mosek.orderingtype.appminloc1`
Approximate minimum local-fill-in ordering is used.

- (2) `mosek.orderingtype.appminloc2`
A variant of the approximate minimum local-fill-in ordering is used.
- (0) `mosek.orderingtype.free`
The ordering method is chosen automatically.
- (3) `mosek.orderingtype.graphpar1`
Graph partitioning based ordering.
- (4) `mosek.orderingtype.graphpar2`
An alternative graph partitioning based ordering.
- (5) `mosek.orderingtype.none`
No ordering is used.

17.32 Parameter type

- (1) `mosek.parametertype.dou_type`
Is a double parameter.
- (2) `mosek.parametertype.int_type`
Is an integer parameter.
- (0) `mosek.parametertype.invalid_type`
Not a valid parameter.
- (3) `mosek.parametertype.str_type`
Is a string parameter.

17.33 Presolve method.

- (2) `mosek.presolvemode.free`
It is decided automatically whether to presolve before the problem is optimized.
- (0) `mosek.presolvemode.off`
The problem is not presolved before it is optimized.
- (1) `mosek.presolvemode.on`
The problem is presolved before it is optimized.

17.34 Problem data items

- (1) `mosek.problemitem.con`
Item is a constraint.

- (2) `mosek.problemitem.cone`
Item is a cone.
- (0) `mosek.problemitem.var`
Item is a variable.

17.35 Problem types

- (4) `mosek.problemttype.conic`
A conic optimization.
- (3) `mosek.problemttype.geco`
General convex optimization.
- (0) `mosek.problemttype.lo`
The problem is a linear optimization problem.
- (5) `mosek.problemttype.mixed`
General nonlinear constraints and conic constraints. This combination can not be solved by MOSEK.
- (2) `mosek.problemttype.qcqp`
The problem is a quadratically constrained optimization problem.
- (1) `mosek.problemttype.qp`
The problem is a quadratic optimization problem.

17.36 Problem status keys

- (3) `mosek.prosta.dual_feas`
The problem is dual feasible.
- (5) `mosek.prosta.dual_infeas`
The problem is dual infeasible.
- (7) `mosek.prosta.ill_posed`
The problem is ill-posed. For example, it may be primal and dual feasible but have a positive duality gap.
- (10) `mosek.prosta.near_dual_feas`
The problem is at least nearly dual feasible.
- (8) `mosek.prosta.near_prim_and_dual_feas`
The problem is at least nearly primal and dual feasible.
- (9) `mosek.prosta.near_prim_feas`
The problem is at least nearly primal feasible.

- (1) `mosek.prosta.prim_and_dual_feas`
The problem is primal and dual feasible.
- (6) `mosek.prosta.prim_and_dual_infeas`
The problem is primal and dual infeasible.
- (2) `mosek.prosta.prim_feas`
The problem is primal feasible.
- (4) `mosek.prosta.prim_infeas`
The problem is primal infeasible.
- (11) `mosek.prosta.prim_infeas_or_unbounded`
The problem is either primal infeasible or unbounded. This may occur for mixed-integer problems.
- (0) `mosek.prosta.unknown`
Unknown problem status.

17.37 Interpretation of quadratic terms in MPS files

- (0) `mosek.qreadtype.add`
All elements in a Q matrix are assumed to belong to the lower triangular part. Duplicate elements in a Q matrix are added together.
- (1) `mosek.qreadtype.drop_lower`
All elements in the strict lower triangular part of the Q matrices are dropped.
- (2) `mosek.qreadtype.drop_upper`
All elements in the strict upper triangular part of the Q matrices are dropped.

17.38 Response code type

- (3) `mosek.rescodetype.err`
The response code is an error.
- (0) `mosek.rescodetype.ok`
The response code is OK.
- (2) `mosek.rescodetype.trm`
The response code is an optimizer termination status.
- (4) `mosek.rescodetype.unk`
The response code does not belong to any class.
- (1) `mosek.rescodetype.wrn`
The response code is a warning.

17.39 Scaling type

- (1) `mosek.scalingmethod.free`
The optimizer chooses the scaling heuristic.
- (0) `mosek.scalingmethod.pow2`
Scales only with power of 2 leaving the mantissa untouched.

17.40 Scaling type

- (3) `mosek.scalingtype.aggressive`
A very aggressive scaling is performed.
- (0) `mosek.scalingtype.free`
The optimizer chooses the scaling heuristic.
- (2) `mosek.scalingtype.moderate`
A conservative scaling is performed.
- (1) `mosek.scalingtype.none`
No scaling is performed.

17.41 Sensitivity types

- (0) `mosek.sensitivitytype.basis`
Basis sensitivity analysis is performed.
- (1) `mosek.sensitivitytype.optimal_partition`
Optimal partition sensitivity analysis is performed.

17.42 Degeneracy strategies

- (2) `mosek.simdegen.aggressive`
The simplex optimizer should use an aggressive degeneration strategy.
- (1) `mosek.simdegen.free`
The simplex optimizer chooses the degeneration strategy.
- (4) `mosek.simdegen.minimum`
The simplex optimizer should use a minimum degeneration strategy.
- (3) `mosek.simdegen.moderate`
The simplex optimizer should use a moderate degeneration strategy.
- (0) `mosek.simdegen.none`
The simplex optimizer should use no degeneration strategy.

17.43 Exploit duplicate columns.

- (2) `mosek.simdupvec.free`
The simplex optimizer can choose freely.
- (0) `mosek.simdupvec.off`
Disallow the simplex optimizer to exploit duplicated columns.
- (1) `mosek.simdupvec.on`
Allow the simplex optimizer to exploit duplicated columns.

17.44 Hot-start type employed by the simplex optimizer

- (1) `mosek.simhotstart.free`
The simplex optimizer chooses the hot-start type.
- (0) `mosek.simhotstart.none`
The simplex optimizer performs a coldstart.
- (2) `mosek.simhotstart.status_keys`
Only the status keys of the constraints and variables are used to choose the type of hot-start.

17.45 Problem reformulation.

- (3) `mosek.simreform.aggressive`
The simplex optimizer should use an aggressive reformulation strategy.
- (2) `mosek.simreform.free`
The simplex optimizer can choose freely.
- (0) `mosek.simreform.off`
Disallow the simplex optimizer to reformulate the problem.
- (1) `mosek.simreform.on`
Allow the simplex optimizer to reformulate the problem.

17.46 Simplex selection strategy

- (2) `mosek.simseltype.ase`
The optimizer uses approximate steepest-edge pricing.
- (3) `mosek.simseltype.devex`
The optimizer uses devex steepest-edge pricing (or if it is not available an approximate steep-edge selection).

- (0) `mosek.simseltype.free`
The optimizer chooses the pricing strategy.
- (1) `mosek.simseltype.full`
The optimizer uses full pricing.
- (5) `mosek.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.
- (4) `mosek.simseltype.se`
The optimizer uses steepest-edge selection (or if it is not available an approximate steep-edge selection).

17.47 Solution items

- (3) `mosek.solitem.slc`
Lagrange multipliers for lower bounds on the constraints.
- (5) `mosek.solitem.slx`
Lagrange multipliers for lower bounds on the variables.
- (7) `mosek.solitem.snx`
Lagrange multipliers corresponding to the conic constraints on the variables.
- (4) `mosek.solitem.suc`
Lagrange multipliers for upper bounds on the constraints.
- (6) `mosek.solitem.sux`
Lagrange multipliers for upper bounds on the variables.
- (0) `mosek.solitem.xc`
Solution for the constraints.
- (1) `mosek.solitem.xx`
Variable solution.
- (2) `mosek.solitem.y`
Lagrange multipliers for equations.

17.48 Solution status keys

- (3) `mosek.solsta.dual_feas`
The solution is dual feasible.
- (6) `mosek.solsta.dual_infeas_cer`
The solution is a certificate of dual infeasibility.

- (14) `mosek.solsta.integer_optimal`
The primal solution is integer optimal.
- (10) `mosek.solsta.near_dual_feas`
The solution is nearly dual feasible.
- (13) `mosek.solsta.near_dual_infeas_cer`
The solution is almost a certificate of dual infeasibility.
- (15) `mosek.solsta.near_integer_optimal`
The primal solution is near integer optimal.
- (8) `mosek.solsta.near_optimal`
The solution is nearly optimal.
- (11) `mosek.solsta.near_prim_and_dual_feas`
The solution is nearly both primal and dual feasible.
- (9) `mosek.solsta.near_prim_feas`
The solution is nearly primal feasible.
- (12) `mosek.solsta.near_prim_infeas_cer`
The solution is almost a certificate of primal infeasibility.
- (1) `mosek.solsta.optimal`
The solution is optimal.
- (4) `mosek.solsta.prim_and_dual_feas`
The solution is both primal and dual feasible.
- (2) `mosek.solsta.prim_feas`
The solution is primal feasible.
- (5) `mosek.solsta.prim_infeas_cer`
The solution is a certificate of primal infeasibility.
- (0) `mosek.solsta.unknown`
Status of the solution is unknown.

17.49 Solution types

- (1) `mosek.soltype.bas`
The basic solution.
- (2) `mosek.soltype.itg`
The integer solution.
- (0) `mosek.soltype.itr`
The interior solution.

17.50 Solve primal or dual form

- (2) `mosek.solveform.dual`
The optimizer should solve the dual problem.
- (0) `mosek.solveform.free`
The optimizer is free to solve either the primal or the dual problem.
- (1) `mosek.solveform.primal`
The optimizer should solve the primal problem.

17.51 Status keys

- (1) `mosek.stakey.bas`
The constraint or variable is in the basis.
- (5) `mosek.stakey.fix`
The constraint or variable is fixed.
- (6) `mosek.stakey.inf`
The constraint or variable is infeasible in the bounds.
- (3) `mosek.stakey.low`
The constraint or variable is at its lower bound.
- (2) `mosek.stakey.supbas`
The constraint or variable is super basic.
- (0) `mosek.stakey.unk`
The status for the constraint or variable is unknown.
- (4) `mosek.stakey.upr`
The constraint or variable is at its upper bound.

17.52 Starting point types

- (2) `mosek.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.
- (0) `mosek.startpointtype.free`
The starting point is chosen automatically.
- (1) `mosek.startpointtype.guess`
The optimizer guesses a starting point.

(3) `mosek.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.

17.53 Stream types

(2) `mosek.streamtype.err`

Error stream. Error messages are written to this stream.

(0) `mosek.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.

(1) `mosek.streamtype.msg`

Message stream. Log information relating to performance and progress of the optimization is written to this stream.

(3) `mosek.streamtype.wrn`

Warning stream. Warning messages are written to this stream.

17.54 Integer values

(20) `mosek.value.license_buffer_length`

The length of a license key buffer.

(1024) `mosek.value.max_str_len`

Maximum string length allowed in MOSEK.

17.55 Variable types

(0) `mosek.variabletype.type_cont`

Is a continuous variable.

(1) `mosek.variabletype.type_int`

Is an integer variable.

17.56 XML writer output mode

(1) `mosek.xmlwriteroutputtype.col`

Write in column order.

(0) `mosek.xmlwriteroutputtype.row`
Write in row order.

Appendix A

Problem analyzer examples

This appendix presents a few examples of the output produced by the problem analyzer described in Section 10.1. The first two problems are taken from the MIPLIB 2003 collection, <http://miplib.zib.de/>.

A.1 air04

Analyzing the problem

Constraints	Bounds	Variables
fixed : all	ranged : all	bin : all

Objective, min cx
range: min |c|: 31.0000 max |c|: 2258.00
distrib: |c| vars
 [31, 100) 176
 [100, 1e+03) 8084
 [1e+03, 2.26e+03] 644

Constraint matrix A has
823 rows (constraints)
8904 columns (variables)
72965 (0.995703%) nonzero entries (coefficients)

Row nonzeros, A_i
range: min A_i: 2 (0.0224618%) max A_i: 368 (4.13297%)
distrib: A_i rows rows% acc%
 2 2 0.24 0.24
 [3, 7] 4 0.49 0.73
 [8, 15] 19 2.31 3.04
 [16, 31] 80 9.72 12.76
 [32, 63] 236 28.68 41.43

[64, 127]	289	35.12	76.55
[128, 255]	186	22.60	99.15
[256, 368]	7	0.85	100.00

Column nonzeros, A|j
 range: min A|j: 2 (0.243013%) max A|j: 15 (1.8226%)
 distrib: A|j cols cols% acc%
 2 118 1.33 1.33
 [3, 7] 2853 32.04 33.37
 [8, 15] 5933 66.63 100.00

A nonzeros, A(ij)
 range: all |A(ij)| = 1.00000

Constraint bounds, lb <= Ax <= ub
 distrib: |b| lbs ubs
 [1, 10] 823 823

Variable bounds, lb <= x <= ub
 distrib: |b| lbs ubs
 0 8904 8904
 [1, 10] 8904

A.2 arki001

Analyzing the problem

Constraints		Bounds		Variables	
lower bd:	82	lower bd:	38	cont:	850
upper bd:	946	fixed :	353	bin :	415
fixed :	20	free :	1	int :	123
		ranged :	996		

Objective, min cx
 range: all |c| in {0.00000, 1.00000}
 distrib: |c| vars
 0 1387
 1 1

Constraint matrix A has
 1048 rows (constraints)
 1388 columns (variables)
 20439 (1.40511%) nonzero entries (coefficients)

Row nonzeros, A_i
 range: min A_i: 1 (0.0720461%) max A_i: 1046 (75.3602%)
 distrib: A_i rows rows% acc%
 1 29 2.77 2.77

2	476	45.42	48.19
[3, 7]	49	4.68	52.86
[8, 15]	56	5.34	58.21
[16, 31]	64	6.11	64.31
[32, 63]	373	35.59	99.90
[1024, 1046]	1	0.10	100.00

Column nonzeros, A|j

range: min A j: 1	(0.0954198%)	max A j: 29	(2.76718%)
-------------------	--------------	-------------	------------

distrib:	A j	cols	cols%	acc%
1	381	27.45	27.45	
2	19	1.37	28.82	
[3, 7]	38	2.74	31.56	
[8, 15]	233	16.79	48.34	
[16, 29]	717	51.66	100.00	

A nonzeros, A(ij)

range: min A(ij) :	0.000200000	max A(ij) :	2.33067e+07
---------------------	-------------	--------------	-------------

distrib:	A(ij)	coeffs
[0.0002, 0.001)	167	
[0.001, 0.01)	1049	
[0.01, 0.1)	4553	
[0.1, 1)	8840	
[1, 10)	3822	
[10, 100)	630	
[100, 1e+03)	267	
[1e+03, 1e+04)	699	
[1e+04, 1e+05)	291	
[1e+05, 1e+06)	83	
[1e+06, 1e+07)	19	
[1e+07, 2.33e+07]	19	

Constraint bounds, $lb \leq Ax \leq ub$

distrib:	b	lbs	ubs
[0.1, 1)			386
[1, 10)			74
[10, 100)	101		456
[100, 1000)			34
[1000, 10000)			15
[10000, 1e+06]	1		1

Variable bounds, $lb \leq x \leq ub$

distrib:	b	lbs	ubs
0	974		323
[0.001, 0.01)			19
[0.1, 1)	370		57
[1, 10)	41		704
[10, 100]	2		246

A.3 Problem with both linear and quadratic constraints

Analyzing the problem

Constraints		Bounds		Variables
lower bd:	40	upper bd:	1	cont: all
upper bd:	121	fixed :	204	
fixed :	5480	free :	5600	
ranged :	161	ranged :	40	

Objective, maximize cx
 range: all |c| in {0.00000, 15.4737}
 distrib: |c| vars
 0 5844
 15.4737 1

Constraint matrix A has
 5802 rows (constraints)
 5845 columns (variables)
 6480 (0.0191079%) nonzero entries (coefficients)

Row nonzeros, A_i
 range: min A_i: 0 (0%) max A_i: 3 (0.0513259%)
 distrib: A_i rows rows% acc%
 0 80 1.38 1.38
 1 5003 86.23 87.61
 2 680 11.72 99.33
 3 39 0.67 100.00

0/80 empty rows have quadratic terms

Column nonzeros, A_j
 range: min A_j: 0 (0%) max A_j: 15 (0.258532%)
 distrib: A_j cols cols% acc%
 0 204 3.49 3.49
 1 5521 94.46 97.95
 2 40 0.68 98.63
 [3, 7] 40 0.68 99.32
 [8, 15] 40 0.68 100.00

0/204 empty columns correspond to variables used in conic
 and/or quadratic expressions only

A nonzeros, A(ij)
 range: min |A(ij)|: 2.02410e-05 max |A(ij)|: 35.8400
 distrib: A(ij) coeffs
 [2.02e-05, 0.0001) 40
 [0.0001, 0.001) 118
 [0.001, 0.01) 305
 [0.01, 0.1) 176
 [0.1, 1) 40
 [1, 10) 5721
 [10, 35.8] 80

Constraint bounds, lb ≤ Ax ≤ ub
 distrib: |b| lbs ub
 0 5481 5600

```

[1000, 10000)                                1
[10000, 100000)                               2           1
[1e+06, 1e+07)                               78           40
[1e+08, 1e+09]                               120          120

Variable bounds, lb <= x <= ub
distrib:    |b|          lbs          ub
            0           243          203
            [0.1, 1)     1           1
            [1e+06, 1e+07)         40
            [1e+11, 1e+12]         1

-----

Quadratic constraints: 121

Gradient nonzeros, Qx
range: min Qx: 1 (0.0171086%)    max Qx: 2720 (46.5355%)
distrib:    Qx          cons      cons%      acc%
            1           40         33.06       33.06
            [64, 127]    80         66.12       99.17
            [2048, 2720]  1          0.83       100.00

-----

```

A.4 Problem with both linear and conic constraints

Analyzing the problem

```

Constraints          Bounds          Variables
upper bd:           3600      fixed   :      3601      cont: all
fixed   :           21760     free    :      28802

-----

Objective, minimize cx
range: all |c| in {0.00000, 1.00000}
distrib:    |c|          vars
            0           32402
            1            1

-----

Constraint matrix A has
25360 rows (constraints)
32403 columns (variables)
93339 (0.0113587%) nonzero entries (coefficients)

Row nonzeros, A_i
range: min A_i: 1 (0.00308613%)    max A_i: 8 (0.0246891%)
distrib:    A_i          rows      rows%      acc%
            1           3600       14.20       14.20
            2          10803       42.60       56.79
            [3, 7]       3995       15.75       72.55
            8           6962       27.45      100.00

```

Column nonzeros, $A|j$
 range: min $A|j$: 0 (0%) max $A|j$: 61 (0.240536%)
 distrib: $A|j$ cols cols% acc%

	0	3602	11.12	11.12
	1	10800	33.33	44.45
	2	7200	22.22	66.67
	[3, 7]	7279	22.46	89.13
	[8, 15]	3521	10.87	100.00
	[32, 61]	1	0.00	100.00

3600/3602 empty columns correspond to variables used in conic
 and/or quadratic constraints only

A nonzeros, $A(ij)$
 range: min $|A(ij)|$: 0.00833333 max $|A(ij)|$: 1.00000
 distrib: $A(ij)$ coeffs

	[0.00833, 0.01)	57280
	[0.01, 0.1)	59
	[0.1, 1]	36000

Constraint bounds, $lb \leq Ax \leq ub$

distrib:	$ b $	lbs	ubs
	0	21760	21760
	[0.1, 1]		3600

Variable bounds, $lb \leq x \leq ub$

distrib:	$ b $	lbs	ubs
	[1, 10]	3601	3601

Rotated quadratic cones: 3600

dim	RQCs
4	3600

Appendix B

The MPS file format

MOSEK supports the standard MPS format with some extensions. For a detailed description of the MPS format the book by Nazareth [18] is a good reference.

B.1 The MPS file format

The version of the MPS format supported by MOSEK allows specification of an optimization problem on the form

$$\begin{aligned} l^c &\leq Ax + q(x) &\leq u^c, \\ l^x &\leq x &\leq u^x, \\ x &\in \mathcal{C}, \\ x_{\mathcal{J}} &\text{ integer}, \end{aligned} \tag{B.1}$$

where

- $x \in \mathbb{R}^n$ is the vector of decision variables.
- $A \in \mathbb{R}^{m \times n}$ is the constraint matrix.
- $l^c \in \mathbb{R}^m$ is the lower limit on the activity for the constraints.
- $u^c \in \mathbb{R}^m$ is the upper limit on the activity for the constraints.
- $l^x \in \mathbb{R}^n$ is the lower limit on the activity for the variables.
- $u^x \in \mathbb{R}^n$ is the upper limit on the activity for the variables.
- $q : \mathbb{R}^n \rightarrow \mathbb{R}$ is a vector of quadratic functions. Hence,

$$q_i(x) = 1/2 x^T Q^i x$$

where it is assumed that

$$Q^i = (Q^i)^T. \tag{B.2}$$

Please note the explicit 1/2 in the quadratic term and that Q^i is required to be symmetric.

- \mathcal{C} is a convex cone.
- $\mathcal{J} \subseteq \{1, 2, \dots, n\}$ is an index set of the integer-constrained variables.

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] [vname3] [value2]
RHS
    [name] [cname1] [value1] [cname2] [value2]
RANGES
    [name] [cname1] [value1] [cname2] [value2]
QSECTION [cname1]
    [vname1] [vname2] [value1] [vname3] [value2]
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 MOSEK specific extensions of 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 Section B.5 for details.

B.1.1 An example

A concrete example of a MPS file is presented below:

```

NAME          EXAMPLE
OBJSENSE
  MIN
ROWS
  N  obj
  L  c1
  L  c2
  L  c3
  L  c4
COLUMNS
  x1      obj      -10.0      c1      0.7
  x1      c2        0.5      c3      1.0
  x1      c4        0.1
  x2      obj      -9.0      c1      1.0
  x2      c2      0.833333333333 c3      0.66666667
  x2      c4        0.25
RHS
  rhs     c1      630.0      c2      600.0
  rhs     c3      708.0      c4      135.0
ENDATA

```

Subsequently each individual section in the MPS format is discussed.

B.1.2 NAME

In this section a name ([name]) is assigned to the problem.

B.1.3 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.

B.1.4 OBJNAME (optional)

This is an optional section that can be used to specify the name of the row that is used as objective function. The OBJNAME section contains one line at most which has the form

objname

objname should be a valid row name.

B.1.5 ROWS

A record in the ROWS section has the form

? [cname1]

where the requirements for the fields are as follows:

Field	Starting position	Maximum width	Required	Description
?	2	1	Yes	Constraint key
[cname1]	5	8	Yes	Constraint name

Hence, in this section each constraint is assigned an 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 the values E, G, L, or N with the following interpretation:

Constraint type	l_i^c	u_i^c
E	finite	l_i^c
G	finite	∞
L	$-\infty$	finite
N	$-\infty$	∞

In the MPS format an 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 .

B.1.6 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	Maximum 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.

B.1.7 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	Maximum 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 type	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.

B.1.8 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	Maximum width	Re- quired	Description
<code>[name]</code>	5	8	Yes	Name of the RANGE vector
<code>[cname1]</code>	15	8	Yes	Constraint name
<code>[value1]</code>	25	12	Yes	Numerical value
<code>[cname2]</code>	40	8	No	Constraint name
<code>[value2]</code>	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			

B.1.9 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 term belongs. A record in the **QSECTION** has the form

[vname1] [vname2] [value1] [vname3] [value2]

where the requirements for each field are:

Field	Starting position	Maximum 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

$$\begin{array}{ll}
 \text{minimize} & -x_2 + 0.5(2x_1^2 - 2x_1x_3 + 0.2x_2^2 + 2x_3^2) \\
 \text{subject to} & x_1 + x_2 + x_3 \geq 1, \\
 & x \geq 0
 \end{array}$$

has the following MPS file representation

```

NAME          qoexp
ROWS
  N  obj
  G  c1
COLUMNS
  x1      c1      1
  x2      obj     -1
  x2      c1      1
  x3      c1      1
RHS
  rhs     c1      1
QSECTION    obj
  x1      x1      2
  x1      x3     -1
  x2      x2      0.2
  x3      x3      2
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 .

B.1.10 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	Maximum 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	Variable name

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 which bounds are modified by the record. ?? and [value1] are used to modify the bound vectors according to the following table:

??	l_j^x	u_j^x	Made integer (added to \mathcal{J})
FR	$-\infty$	∞	No
FX	v_1	v_1	No
LO	v_1	unchanged	No
MI	$-\infty$	unchanged	No
PL	unchanged	∞	No
UP	unchanged	v_1	No
BV	0	1	Yes
LI	$[v_1]$	∞	Yes
UI	unchanged	$[v_1]$	Yes

v_1 is the value specified by [value1].

B.1.11 CSECTION (optional)

The purpose of the CSECTION is to specify the constraint

$$x \in \mathcal{C}.$$

in (B.1).

It is assumed that \mathcal{C} 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

$$\mathcal{C} := \{x \in \mathbb{R}^n : x^t \in \mathcal{C}_t, \quad t = 1, \dots, k\}$$

where \mathcal{C}_t must have one of the following forms

- \mathbb{R} set:

$$\mathcal{C}_t = \{x \in \mathbb{R}^{n^t}\}.$$

- Quadratic cone:

$$\mathcal{C}_t = \left\{ x \in \mathbb{R}^{n^t} : x_1 \geq \sqrt{\sum_{j=2}^{n^t} x_j^2} \right\}. \quad (\text{B.3})$$

- Rotated quadratic cone:

$$\mathcal{C}_t = \left\{ x \in \mathbb{R}^{n^t} : 2x_1x_2 \geq \sum_{j=3}^{n^t} x_j^2, \quad x_1, x_2 \geq 0 \right\}. \quad (\text{B.4})$$

In general, only quadratic and rotated quadratic cones are specified in the MPS file whereas membership of the \mathbb{R} set is not. If a variable is not a member of any other cone then it is assumed to be a member of an \mathbb{R} cone.

Next, let us study an example. Assume that the quadratic cone

$$x_4 \geq \sqrt{x_5^2 + x_6^2} \quad (\text{B.5})$$

and the rotated quadratic cone

$$2x_3x_7 \geq x_1^2 + x_8^2, \quad x_3, x_7 \geq 0, \quad (\text{B.6})$$

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
*23456789012345678901234567890123456789012345678901234567890
CSECTION      konea      0.0      QUAD
      x4
      x5
      x8
CSECTION      koneb      0.0      RQUAD
      x7
      x3
      x1
      x0

```

This first CSECTION specifies the cone (B.5) which is given the name **konea**. This is a quadratic cone which is specified by the keyword **QUAD** in the CSECTION header. The 0.0 value in the CSECTION header is not used by the **QUAD** cone.

The second CSECTION specifies the rotated quadratic cone (B.6). Please note the keyword **RQUAD** in the CSECTION which is used to specify that the cone is a rotated quadratic cone instead of a quadratic cone. The 0.0 value in the CSECTION header is not used by the **RQUAD** cone.

In general, a CSECTION header has the format

```
CSECTION      [kname1]      [value1]      [ktype]
```

where the requirement for each field are as follows:

Field	Starting position	Maximum width	Required	Description
[kname1]	5	8	Yes	Name of the cone
[value1]	15	12	No	Cone parameter
[ktype]	25		Yes	Type of the cone.

The possible cone type keys are:

Cone type key	Members	Interpretation.
QUAD	≥ 1	Quadratic cone i.e. (B.3).
RQUAD	≥ 2	Rotated quadratic cone i.e. (B.4).

Please note that a quadratic cone must have at least one member whereas a rotated quadratic cone must have at least two members. A record in the CSECTION has the format

```
[vname1]
```

where the requirements for each field are

Field	Starting position	Maximum width	Required	Description
[vname1]	2	8	Yes	A valid variable name

The most important restriction with respect to the `CSECTION` is that a variable must occur in only one `CSECTION`.

B.1.12 ENDATA

This keyword denotes the end of the MPS file.

B.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 compability 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

- **IMPORTANT:** 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.

B.3 General limitations

- An MPS file should be an ASCII file.

B.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.

B.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, it also presents two main limitations:

- By default a line in the MPS file must not contain more than 1024 characters. However, by modifying the parameter `mosek.iparam.read_mps_width` an arbitrary large line width will be accepted.
- A name must not contain any blanks.

To use the free MPS format instead of the default MPS format the MOSEK parameter `mosek.iparam.read_mps_format` should be changed.

Appendix C

The LP file format

MOSEK supports the LP file format with some extensions i.e. MOSEK can read and write LP formatted files.

C.1 A warning

The LP format is not a well-defined standard and hence different optimization packages may interpret a specific LP formatted file differently.

C.2 The LP file format

The LP file format can specify problems on the form

$$\begin{array}{llll} \text{minimize/maximize} & & c^T x + \frac{1}{2} q^o(x) & \\ \text{subject to} & l^c \leq & Ax + \frac{1}{2} q(x) & \leq u^c, \\ & l^x \leq & x & \leq u^x, \\ & & x_{\mathcal{J}} \text{ integer,} & \end{array}$$

where

- $x \in \mathbb{R}^n$ is the vector of decision variables.
- $c \in \mathbb{R}^n$ is the linear term in the objective.
- $q^o : \mathbb{R}^n \rightarrow \mathbb{R}$ is the quadratic term in the objective where

$$q^o(x) = x^T Q^o x$$

and it is assumed that

$$Q^o = (Q^o)^T. \tag{C.1}$$

- $A \in \mathbb{R}^{m \times n}$ 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

$$Q^i = (Q^i)^T. \quad (\text{C.2})$$

- $\mathcal{J} \subseteq \{1, 2, \dots, n\}$ is an index set of the integer constrained variables.

C.2.1 The 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.

C.2.1.1 The objective

The first section beginning with one of the keywords

```
max
maximum
maximize
min
minimum
minimize
```

defines the objective sense and the objective function, i.e.

$$c^T x + \frac{1}{2} x^T Q^o x.$$

The objective may be given a name by writing

```
myname:
```

before the expressions. 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 in the example

```
4 x1 + x2 - 0.1 x3
```

and so forth. The quadratic terms are written in square brackets ([]) 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 $4 x_1 + 2 x_1$ is equivalent to $6 x_1$. In the quadratic expressions $x_1 * x_2$ is equivalent to $x_2 * x_1$ and as in the linear part, if the same variables multiplied or squared occur several times their coefficients are added.

C.2.1.2 The 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, but MOSEK supports defining ranged constraints by using double-colon (‘‘::’’) instead of a single-colon (‘:’) after the constraint name, i.e.

$$-5 \leq x_1 + x_2 \leq 5 \tag{C.3}$$

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 (C.3) may be written as

$$x_1 + x_2 - sl_1 = 0, \quad -5 \leq sl_1 \leq 5.$$

C.2.1.3 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
```

C.2.1.4 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.

C.2.1.5 Terminating section

Finally, an LP formatted file must be terminated with the keyword

```
end
```

C.2.1.6 An example

A simple example of an LP file with two variables, four constraints and one integer variable is:

```
minimize
  -10 x1 -9 x2
subject to
  0.7 x1 +      x2 <= 630
  0.5 x1 + 0.833 x2 <= 600
      x1 + 0.667 x2 <= 708
  0.1 x1 + 0.025 x2 <= 135
bounds
  10 <= x1
  x1 <= +inf
  20 <= x2 <= 500
general
  x1
end
```

C.2.2 LP format peculiarities

C.2.2.1 Comments

Anything on a line after a “\” is ignored and is treated as a comment.

C.2.2.2 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.

It is strongly recommended not to use double quotes (") in names.

C.2.2.3 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.

C.2.2.4 MOSEK specific 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.

If an LP formatted file created by MOSEK should satisfies the strict definition, then the parameter

`mosek.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.

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).

Internally in MOSEK names may contain any (printable) character, many of which cannot be used in LP names. Setting the parameters

`mosek.iparam.read_lp_quoted_names`

and

`mosek.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.

C.2.3 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 use the parameter setting

```
MSK_IPAR_WRITE_LP_STRICT_FORMAT MSK_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 use the parameter setting

```
MSK_IPAR_WRITE_GENERIC_NAMES MSK_ON
```

which will cause all names to be renamed systematically in the output file.

C.2.4 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

```
MSK_IPAR_WRITE_LP_LINE_WIDTH  
MSK_IPAR_WRITE_LP_TERMS_PER_LINE
```

The first parameter 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.

The second parameter 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.

C.2.4.1 Speeding up file reading

If the input file should be read as fast as possible using the least amount of memory, then it is important to tell MOSEK how many non-zeros, variables and constraints the problem contains. These values can be set using the parameters

```
MSK_IPAR_READ_CON
```

MSK_IPAR_READ_VAR
MSK_IPAR_READ_ANZ
MSK_IPAR_READ_QNZ

C.2.4.2 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).

Appendix D

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.

D.1 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 to; 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.

D.2 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 in quotes [/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]

```

D.2.1 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 the subsection ‘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 the subsections ‘b’ (linear bounds on variables) and ‘cone’ (quadratic cone).
 - **[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

$$-10 \leq x, y \leq 10. \quad (\text{D.1})$$

- **[cone]**. Currently, the supported cones are the *quadratic cone* and the *rotated quadratic cone* (see section 5.4). A conic constraint is defined as a set of variables which belongs to a single unique cone.

A quadratic cone of n variables x_1, \dots, x_n defines a constraint of the form

$$x_1^2 > \sum_{i=2}^n x_i^2.$$

A rotated quadratic cone of n variables x_1, \dots, x_n defines a constraint of the form

$$x_1 x_2 > \sum_{i=3}^n x_i^2.$$

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 quad]  x,y,z,w  [/cone] # quadratic cone
[cone rquad] x,y,z,w  [/cone] # rotated quadratic cone
[/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 values.
- **[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 in a subsection is defined as follows:

```
[hint ITEM] value [/hint]
```

where **ITEM** may be replaced by **numvar** (number of variables), **numcon** (number of linear/quadratic constraints), **numanz** (number of linear non-zeros in constraints) and **numqnz** (number of quadratic non-zeros in constraints).

- **[solutions]** This section can contain a number of full or partial solutions to a problem, each inside a **[solution]**-section. The syntax is

```
[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 values for a single variable or constraint, each value written as

KEYWORD=value

where KEYWORD defines a solution item and value defines its 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 variable associated with its lower bound.
- **su** Defines the level of the 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** and **lv1**, and optionally **s1**, **su** and **sn**.

A [con] section should always contain **sk** and **lv1**, and optionally **s1**, **su** and **y**.

An example of a solution section

```
[solution basic status=UNKNOWN]
  [var x0] sk=LOW    lvl=5.0      [/var]
  [var x1] sk=UPR    lvl=10.0     [/var]
  [var x2] sk=SUPBAS lvl=2.0    sl=1.5 su=0.0 [/var]

  [con c0] sk=LOW    lvl=3.0 y=0.0 [/con]
  [con c0] sk=UPR    lvl=0.0 y=5.0 [/con]
[/solution]
```

- **[vendor]** This contains solver/vendor specific data. It accepts one argument, which is a vendor ID – for MOSEK the ID is simply `mosek` – and the section contains the subsection **parameters** defining solver parameters. When reading a vendor section, any unknown vendor can be safely ignored. This is described later.

Comments using the ‘#’ may appear anywhere in the file. Between the ‘#’ and the following line-break any text may be written, including markup characters.

D.2.2 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]+)?
```

D.2.3 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"
```

D.3 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]
```

D.4 Writing OPF files from MOSEK

The function `Task.writedata` can be used to produce an OPF file from a task.

To write an OPF file set the parameter `mosek.iparam.write_data_format` to `mosek.dataformat.opf` as this ensures that OPF format is used. Then modify the following parameters to define what the file should contain:

- `mosek.iparam.opf_write_header`, include a small header with comments.
- `mosek.iparam.opf_write_hints`, include hints about the size of the problem.
- `mosek.iparam.opf_write_problem`, include the problem itself — objective, constraints and bounds.

- `mosek.iparam.opf_write_solutions`, include solutions if they are defined. If this is off, no solutions are included.
- `mosek.iparam.opf_write_sol_bas`, include basic solution, if defined.
- `mosek.iparam.opf_write_sol_itg`, include integer solution, if defined.
- `mosek.iparam.opf_write_sol_itr`, include interior solution, if defined.
- `mosek.iparam.opf_write_parameters`, include all parameter settings.

D.5 Examples

This section contains a set of small examples written in OPF and describing how to formulate linear, quadratic and conic problems.

D.5.1 Linear example lo1.opf

Consider the example:

$$\begin{aligned}
 &\text{minimize} && -10x_1 && -9x_2, \\
 &\text{subject to} && 7/10x_1 + 1x_2 &\leq 630, \\
 & && 1/2x_1 + 5/6x_2 &\leq 600, \\
 & && 1x_1 + 2/3x_2 &\leq 708, \\
 & && 1/10x_1 + 1/4x_2 &\leq 135, \\
 & && x_1, && x_2 &\geq 0.
 \end{aligned} \tag{D.2}$$

In the OPF format the example is displayed as shown below:

```

1  [comment]
2    Example lo1.mps converted to OPF.
3  [/comment]
4
5  [hints]
6    # Give a hint about the size of the different elements in the problem.
7    # These need only be estimates, but in this case they are exact.
8    [hint NUMVAR] 2 [/hint]
9    [hint NUMCON] 4 [/hint]
10   [hint NUMANZ] 8 [/hint]
11 [/hints]
12
13 [variables]
14   # All variables that will appear in the problem
15   x1 x2
16 [/variables]
17
18 [objective minimize 'obj']
19   - 10 x1 - 9 x2
20 [/objective]
21
22 [constraints]
23   [con 'c1'] 0.7 x1 +          x2 <= 630 [/con]

```

```

24 [con 'c2'] 0.5 x1 + 0.8333333333 x2 <= 600 [/con]
25 [con 'c3']      x1 + 0.666666667 x2 <= 708 [/con]
26 [con 'c4'] 0.1 x1 + 0.25      x2 <= 135 [/con]
27 [/constraints]
28
29 [bounds]
30 # By default all variables are free. The following line will
31 # change this to all variables being nonnegative.
32 [b] 0 <= * [/b]
33 [/bounds]

```

D.5.2 Quadratic example qo1.opf

An example of a quadratic optimization problem is

$$\begin{aligned}
 & \text{minimize} && x_1^2 + 0.1x_2^2 + x_3^2 - x_1x_3 - x_2 \\
 & \text{subject to} && 1 \leq x_1 + x_2 + x_3, \\
 & && x \geq 0.
 \end{aligned} \tag{D.3}$$

This can be formulated in `opf` as shown below.

```

1 [comment]
2   Example qo1.mps converted to OPF.
3 [/comment]
4
5 [hints]
6   [hint NUMVAR] 3 [/hint]
7   [hint NUMCON] 1 [/hint]
8   [hint NUMANZ] 3 [/hint]
9 [/hints]
10
11 [variables]
12   x1 x2 x3
13 [/variables]
14
15 [objective minimize 'obj']
16   # The quadratic terms are often multiplied by 1/2,
17   # but this is not required.
18
19   - x2 + 0.5 ( 2 x1 ^ 2 - 2 x3 * x1 + 0.2 x2 ^ 2 + 2 x3 ^ 2 )
20 [/objective]
21
22 [constraints]
23   [con 'c1'] 1 <= x1 + x2 + x3 [/con]
24 [/constraints]
25
26 [bounds]
27   [b] 0 <= * [/b]
28 [/bounds]

```

D.5.3 Conic quadratic example cqo1.opf

Consider the example:

$$\begin{aligned}
 &\text{minimize} && 1x_1 + 2x_2 \\
 &\text{subject to} && 2x_3 + 4x_4 = 5, \\
 & && x_5^2 \leq 2x_1x_3, \\
 & && x_6^2 \leq 2x_2x_4, \\
 & && x_5 = 1, \\
 & && x_6 = 1, \\
 & && x \geq 0.
 \end{aligned} \tag{D.4}$$

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.

```

1  [comment]
2    Example cqo1.mps converted to OPF.
3  [/comment]
4
5  [hints]
6    [hint NUMVAR] 6 [/hint]
7    [hint NUMCON] 1 [/hint]
8    [hint NUMANZ] 2 [/hint]
9  [/hints]
10
11 [variables]
12   x1 x2 x3 x4 x5 x6
13 [/variables]
14
15 [objective minimize 'obj']
16   x1 + 2 x2
17 [/objective]
18
19 [constraints]
20   [con 'c1'] 2 x3 + 4 x4 = 5 [/con]
21 [/constraints]
22
23 [bounds]
24   # We let all variables default to the positive orthant
25   [b] 0 <= * [/b]
26   # ... and change those that differ from the default.
27   [b] x5,x6 = 1 [/b]
28
29   # We define two rotated quadratic cones
30
31   # k1: 2 x1 * x3 >= x5^2
32   [cone rquad 'k1'] x1, x3, x5 [/cone]
33
34   # k2: 2 x2 * x4 >= x6^2
35   [cone rquad 'k2'] x2, x4, x6 [/cone]
36 [/bounds]

```

D.5.4 Mixed integer example milo1.opf

Consider the mixed integer problem:

$$\begin{aligned}
 & \text{maximize} && x_0 + 0.64x_1 \\
 & \text{subject to} && 50x_0 + 31x_1 \leq 250, \\
 & && 3x_0 - 2x_1 \geq -4, \\
 & && x_0, x_1 \geq 0 \quad \text{and integer}
 \end{aligned}
 \tag{D.5}$$

This can be implemented in OPF with:

```

1  [comment]
2    Written by MOSEK version 5.0.0.7
3    Date 20-11-06
4    Time 14:42:24
5  [/comment]
6
7  [hints]
8    [hint NUMVAR] 2 [/hint]
9    [hint NUMCON] 2 [/hint]
10   [hint NUMANZ] 4 [/hint]
11 [/hints]
12
13 [variables disallow_new_variables]
14   x1 x2
15 [/variables]
16
17 [objective maximize 'obj']
18   x1 + 6.4e-1 x2
19 [/objective]
20
21 [constraints]
22   [con 'c1']          5e+1 x1 + 3.1e+1 x2 <= 2.5e+2 [/con]
23   [con 'c2'] -4 <= 3 x1 - 2 x2 [/con]
24 [/constraints]
25
26 [bounds]
27   [b] 0 <= * [/b]
28 [/bounds]
29
30 [integer]
31   x1 x2
32 [/integer]

```


Appendix E

The XML (OSiL) format

MOSEK can write data in the standard OSiL xml format. For a definition of the OSiL format please see <http://www.optimizationservices.org/>. Only linear constraints (possibly with integer variables) are supported. By default output files with the extension `.xml` are written in the OSiL format.

The parameter `mosek.iparam.write_xml_mode` controls if the linear coefficients in the A matrix are written in row or column order.

Appendix F

The ORD file format

An ORD formatted file specifies in which order the mixed integer optimizer branches on variables. The format of an ORD file is shown in Figure F.1. In the figure names in capitals are keywords of the ORD format, whereas names in brackets are custom names or values. The ?? is an optional key specifying the preferred branching direction. The possible keys are DN and UP which indicate that down or up is the preferred branching direction respectively. The branching direction key is optional and is left blank the mixed integer optimizer will decide whether to branch up or down.

```
*           1           2           3           4           5           6
*23456789012345678901234567890123456789012345678901234567890
NAME           [name]
  ?? [vname1]           [value1]
ENDATA
```

Figure F.1: The standard ORD format.

F.1 An example

A concrete example of a ORD file is presented below:

```
NAME           EXAMPLE
  DN x1           2
  UP x2           1
    x3           10
ENDATA
```

This implies that the priorities 2, 1, and 10 are assigned to variable **x1**, **x2**, and **x3** respectively. The higher the priority value assigned to a variable the earlier the mixed integer optimizer will branch on that variable. The key **DN** implies that the mixed integer optimizer first will branch down on variable whereas the key **UP** implies that the mixed integer optimizer will first branch up on a variable.

If no branch direction is specified for a variable then the mixed integer optimizer will automatically choose the branching direction for that variable. Similarly, if no priority is assigned to a variable then it is automatically assigned the priority of 0.

Appendix G

The solution file format

MOSEK provides one or two solution files depending on the problem type and the optimizer used. If a problem is optimized using the interior-point optimizer and no basis identification is required, then a file named **prodbname.sol** is provided. **prodbname** is the name of the problem and **.sol** is the file extension. If the problem is optimized using the simplex optimizer or basis identification is performed, then a file named **prodbname.bas** is created presenting the optimal basis solution. Finally, if the problem contains integer constrained variables then a file named **prodbname.int** is created. It contains the integer solution.

G.1 The basic and interior solution files

In general both the interior-point and the basis solution files have the format:

```
NAME : <problem name>
PROBLEM STATUS : <status of the problem>
SOLUTION STATUS : <status of the solution>
OBJECTIVE NAME : <name of the objective function>
PRIMAL OBJECTIVE : <primal objective value corresponding to the solution>
DUAL OBJECTIVE : <dual objective value corresponding to the solution>
CONSTRAINTS
INDEX NAME AT ACTIVITY LOWER LIMIT UPPER LIMIT DUAL LOWER DUAL UPPER
? <name> ?? <a value> <a value> <a value> <a value> <a value>
VARIABLES
INDEX NAME AT ACTIVITY LOWER LIMIT UPPER LIMIT DUAL LOWER DUAL UPPER CONIC DUAL
? <name> ?? <a value> <a value> <a value> <a value> <a value> <a value>
```

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 statuses are shown. In this case the information shows that the problem is primal and dual feasible and the solution is optimal. Next the primal and dual objective values are displayed.

CONSTRAINTS Subsequently in the constraint section the following information is listed for each constraint:

INDEX A sequential index assigned to the constraint by MOSEK.

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.

Table G.1: Status keys.

NAME The name of the constraint assigned by the user.

AT The status of the constraint. In Table G.1 the possible values of the status keys and their interpretation are shown.

ACTIVITY Given the i th constraint on the form

$$l_i^c \leq \sum_{j=1}^n a_{ij}x_j \leq u_i^c, \quad (\text{G.1})$$

then activity denote the quantity $\sum_{j=1}^n a_{ij}x_j^*$, where x^* is the value for the x solution.

LOWER LIMIT Is the quantity l_i^c (see (G.1)).

UPPER LIMIT Is the quantity u_i^c (see (G.1)).

DUAL LOWER Is the dual multiplier corresponding to the lower limit on the constraint.

DUAL UPPER Is the dual multiplier corresponding to the upper limit on the constraint.

VARIABLES The last section of the solution report lists information for the variables. This information has a similar interpretation as for the constraints. However, the column with the header [CONIC DUAL] is only included for problems having one or more conic constraints. This column shows the dual variables corresponding to the conic constraints.

G.2 The integer solution file

The integer solution is equivalent to the basic and interior solution files except that no dual information is included.

Bibliography

- [1] Richard C. Grinold and Ronald N. Kahn. *Active portfolio management*. McGraw-Hill, New York, 2 edition, 2000.
- [2] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. Network flows. In G. L. Nemhauser, A. H. G. Rinnooy Kan, and M. J. Todd, editors, *Optimization*, volume 1, pages 211–369. North Holland, Amsterdam, 1989.
- [3] F. Alizadeh and D. Goldfarb. Second-order cone programming. *Math. Programming*, 95(1):3–51, 2003.
- [4] E. D. Andersen and K. D. Andersen. Presolving in linear programming. *Math. Programming*, 71(2):221–245, 1995.
- [5] 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.
- [6] E. D. Andersen, C. Roos, and T. Terlaky. On implementing a primal-dual interior-point method for conic quadratic optimization. *Math. Programming*, 95(2), February 2003.
- [7] E. D. Andersen and Y. Ye. Combining interior-point and pivoting algorithms. *Management Sci.*, 42(12):1719–1731, December 1996.
- [8] E. D. Andersen and Y. Ye. A computational study of the homogeneous algorithm for large-scale convex optimization. *Computational Optimization and Applications*, 10:243–269, 1998.
- [9] E. D. Andersen and Y. Ye. On a homogeneous algorithm for the monotone complementarity problem. *Math. Programming*, 84(2):375–399, February 1999.
- [10] Erling D. Andersen. The homogeneous and self-dual model and algorithm for linear optimization. Technical Report TR-1-2009, MOSEK ApS, 2009. <http://www.mosek.com/fileadmin/reports/tech/homolo.pdf>.
- [11] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty. *Nonlinear programming: Theory and algorithms*. John Wiley and Sons, New York, 2 edition, 1993.
- [12] A. Ben-Tal and A. Nemirovski. *Lectures on Modern Convex Optimization: Analysis, Algorithms, and Engineering Applications*. MPS/SIAM Series on Optimization. SIAM, 2001.

- [13] V. Chvátal. *Linear programming*. W.H. Freeman and Company, 1983.
- [14] N. Gould and P. L. Toint. Preprocessing for quadratic programming. *Math. Programming*, 100(1):95–132, 2004.
- [15] J. L. Kenningon and K. R. Lewis. Generalized networks: The theory of preprocessing and an empirical analysis. *INFORMS Journal on Computing*, 16(2):162–173, 2004.
- [16] M. S. Lobo, L. Vanderberghe, S. Boyd, and H. Lebret. Applications of second-order cone programming. *Linear Algebra Appl.*, 284:193–228, November 1998.
- [17] M. S. Lobo and M. Fazel, and S. Boyd. Portfolio optimization with linear and fixed transaction costs. Technical report, CDS, California Institute of Technology, 2005. To appear in *Annals of Operations Research*. <http://www.cds.caltech.edu/~maryam/portfolio.html>.
- [18] J. L. Nazareth. *Computer Solution of Linear Programs*. Oxford University Press, New York, 1987.
- [19] 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.
- [20] Bernd Scherer. *Portfolio construction and risk budgeting*. Risk Books, 2 edition, 2004.
- [21] G. W. Stewart. *Matrix Algorithms. Volume 1: Basic decompositions*. SIAM, 1998.
- [22] S. W. Wallace. Decision making under uncertainty: Is sensitivity of any use. *Oper. Res.*, 48(1):20–25, January 2000.
- [23] H. P. Williams. *Model building in mathematical programming*. John Wiley and Sons, 3 edition, 1993.
- [24] L. A. Wolsey. *Integer programming*. John Wiley and Sons, 1998.

Index

- absolute value, 88
- `mosek.accmode`, 412
- `mosek.adopcode`, 412
- `mosek.adoptype`, 413
- `alloc_add_qnz` (parameter), 313
- `ana_sol_basis` (parameter), 313
- `ana_sol_infeas_tol` (parameter), 282
- `ana_sol_print_violated` (parameter), 314
- `analyzeproblem` (Task method), 188
- `analyzesolution` (Task method), 188
- `append` (Task method), 188
- `appendcone`
 - Task.appendcone (example), 37
- `appendcone` (Task method), 189
- attaching streams, 23
- `auto_sort_a_before_opt` (parameter), 314
- `auto_update_sol_info` (parameter), 314

- `bas_sol_file_name` (parameter), 378
- `mosek.basindtype`, 413
- basis identification, 99
- `basis_rel_tol_s` (parameter), 283
- `basis_solve_use_plus_one` (parameter), 315
- `basis_tol_s` (parameter), 283
- `basis_tol_x` (parameter), 283
- `basiscond` (Task method), 190
- `bi_clean_optimizer` (parameter), 315
- `bi_ignore_max_iter` (parameter), 316
- `bi_ignore_num_error` (parameter), 316
- `bi_max_iterations` (parameter), 316
- `mosek.boundkey`, 413
- bounds, infinite, 72
- `mosek.branchdir`, 414

- `cache_license` (parameter), 316
- `cache_size_l1` (parameter), 317
- `cache_size_l2` (parameter), 317
- `callback_freq` (parameter), 283

- `mosek.callbackcode`, 414
- certificate
 - dual, 74
 - primal, 73
- `check_convexity` (parameter), 317
- `check_convexity_rel_tol` (parameter), 284
- `check_task_data` (parameter), 318
- `checkconvexity` (Task method), 190
- `mosek.checkconvexitytype`, 422
- `checkdata` (Task method), 190
- `checkinlicense` (Env method), 174
- `checkmemtask` (Task method), 190
- `checkoutlicense` (Env method), 174
- `chgbound` (Task method), 191
- `commitchanges` (Task method), 191
- complementarity conditions, 73
- `mosek.compresstype`, 422
- concurrent
 - `mosek.optimizertype.concurrent` (example), 107
- concurrent optimization, 106
- concurrent solution, 105
- concurrent-num-optimizers
 - `mosek.iparam.concurrent_num_optimizers` (example), 107
- `concurrent_num_optimizers` (parameter), 318
- `concurrent_priority_dual_simplex` (parameter), 318
- `concurrent_priority_free_simplex` (parameter), 318
- `concurrent_priority_intpnt` (parameter), 319
- `concurrent_priority_primal_simplex` (parameter), 319
- `mosek.conetype`, 422
- conic, 36
 - optimization, 77
 - problem, 77
- conic modelling, 78

- minimizing norms, example, 80
 - pitfalls, 85
 - quadratic objective, example, 79
 - risk and market impact, example
 - Markowitz model, example, 88
- conic optimization, 36
- conic problem example, 37
- conic quadratic optimization, 36
- constraint
 - matrix, 71, 457
 - quadratic, 75
- constraints
 - lower limit, 72, 457
 - number of, 18
 - upper limit, 72, 457
- continuous relaxation, 113
- convex quadratic problem, 28
- cpu_type (parameter), 319
- mosek.cpu_type, 422
- data_check (parameter), 320
- data_file_name (parameter), 378
- data_tol_ajj (parameter), 284
- data_tol_ajj_huge (parameter), 284
- data_tol_ajj_large (parameter), 285
- data_tol_bound_inf (parameter), 285
- data_tol_bound_wrn (parameter), 285
- data_tol_c_huge (parameter), 285
- data_tol_cj_large (parameter), 286
- data_tol_qij (parameter), 286
- data_tol_x (parameter), 286
- mosek.dataformat, 423
- debug_file_name (parameter), 379
- deletesolution (Task method), 191
- mosek.dinfitem, 424
- dual certificate, 74
- dual infeasible, 72, 74
- duality gap (linear problem), 73
- dualizer, 95
- dualsensitivity
 - Task.dualsensitivity (example), 146
- dualsensitivity (Task method), 192
- echointro (Env method), 174
- eliminator, 94
- Embedded network flow problems, 65, 103
- Env
 - constructors, 173
 - env, creating, 16
 - env, initializing, 16
 - Env.Env (constructor), 173
 - Error
 - constructors, 177
 - Error.Error (constructor), 177
 - example
 - conic problem, 37
 - cqo1.py, 37
 - lo1, 19
 - lo2, 25
 - lo2.python, 25
 - milol.py, 41
 - miainitsol.py, 44
 - network1, 62
 - network2, 66
 - qo1.py, 29
 - qo1, 29
 - quadratic constraints, 33
 - quadratic objective, 29
 - simple, 16
 - Exception
 - constructors, 178
 - Exception.Exception (constructor), 178
 - executing Python script (Linux), 13
 - feasible, primal, 72
 - feasrepair-optimize
 - mosek.iparam.feasrepair_optimize (example), 136
 - feasrepair_name_prefix (parameter), 379
 - feasrepair_name_separator (parameter), 379
 - feasrepair_name_wsumviol (parameter), 379
 - feasrepair_optimize (parameter), 320
 - feasrepair_tol (parameter), 287
 - mosek.feasrepairtype, 428
 - mosek.feature, 428
 - getaij (Task method), 192
 - getapieceumnz (Task method), 193
 - getaslice64 (Task method), 193
 - getaslicenumnz64 (Task method), 194
 - getaslicetrip (Task method), 194
 - getavec (Task method), 195
 - getavecnunz (Task method), 195
 - getbound (Task method), 195

- getboundslice (Task method), 196
- getc (Task method), 196
- getcfix (Task method), 196
- getcone (Task method), 197
- getconeinfo (Task method), 197
- getconname64 (Task method), 197
- getcslice (Task method), 197
- getdbi (Task method), 198
- getdcni (Task method), 198
- getdeqi (Task method), 199
- getdouinf (Task method), 199
- getdoupparam (Task method), 199
- getdualobj (Task method), 199
- getinfeasiblesubproblem (Task method), 200
- getinti (Task method), 200
- getintinf (Task method), 200
- getintparam (Task method), 201
- getintpntnumthreads (Task method), 201
- getlintinf (Task method), 201
- getmaxnumanz64 (Task method), 201
- getmaxnumcon (Task method), 201
- getmaxnumcone (Task method), 202
- getmaxnumqnz64 (Task method), 202
- getmaxnumvar (Task method), 202
- getmemusagetask64 (Task method), 202
- getname64 (Task method), 202
- getnameapi64 (Task method), 203
- getnameindex (Task method), 203
- getnamelen64 (Task method), 203
- getnumanz (Task method), 204
- getnumanz64 (Task method), 204
- getnumcon (Task method), 204
- getnumcone (Task method), 204
- getnumconemem (Task method), 204
- getnumintvar (Task method), 204
- getnumparam (Task method), 205
- getnumqconknz (Task method), 205
- getnumqconknz64 (Task method), 205
- getnumqobjnz (Task method), 205
- getnumqobjnz64 (Task method), 205
- getnumvar (Task method), 205
- getobjname64 (Task method), 206
- getobjsense (Task method), 206
- getpbi (Task method), 206
- getpcni (Task method), 207
- getpeqi (Task method), 207
- getprimalobj
 - Task.getprimalobj (example), 136
- getprimalobj (Task method), 207
- getprobtype (Task method), 208
- getqconk (Task method), 208
- getqconk64 (Task method), 208
- getqobj (Task method), 209
- getqobj64 (Task method), 209
- getqobjij (Task method), 210
- getreducedcosts (Task method), 210
- getsolution (Task method), 210
- getsolutioni (Task method), 212
- getsolutioninf (Task method), 212
- getsolutionslice (Task method), 214
- getsolutionstatus (Task method), 215
- getsolutionstatuskeyslice (Task method), 215
- gettaskname64 (Task method), 216
- getvarbranchdir (Task method), 216
- getvarbranchpri (Task method), 216
- getvarname64 (Task method), 217
- getvartype (Task method), 217
- getvartypelist (Task method), 217
- getversion (Env method), 174
- help desk, 9
- hot-start, 101
- mosek.iinfitem**, 428
- infeas_generic_names (parameter), 320
- infeas_prefer_primal (parameter), 321
- infeas_report_auto (parameter), 321
- infeas_report_level (parameter), 321
- infeasible, 123
 - dual, 74
 - primal, 73
- infeasible problems, 123
- infeasible, dual, 72
- infeasible, primal, 72
- infinite bounds, 72
- mosek.inftype**, 435
- initbasissolve (Task method), 217
- initenv
 - Env.initenv (example), 16
- initenv (Env method), 175
- inputdata
 - Task.inputdata (example), 146
- inputdata64 (Task method), 218

- installation, 11
- int_sol_file_name (parameter), 380
- integer optimization, 40, 113
 - relaxation, 113
- interior-point optimizer, 96, 104
- interior-point or simplex optimizer, 102
- intpnt_basis (parameter), 321
- intpnt_co_tol_dfeas (parameter), 287
- intpnt_co_tol_infeas (parameter), 287
- intpnt_co_tol_mu_red (parameter), 287
- intpnt_co_tol_near_rel (parameter), 288
- intpnt_co_tol_pfeas (parameter), 288
- intpnt_co_tol_rel_gap (parameter), 288
- intpnt_diff_step (parameter), 322
- intpnt_factor_debug_lvl (parameter), 322
- intpnt_factor_method (parameter), 323
- intpnt_max_iterations (parameter), 323
- intpnt_max_num_cor (parameter), 323
- intpnt_max_num_refinement_steps (parameter), 323
- intpnt_nl_merit_bal (parameter), 289
- intpnt_nl_tol_dfeas (parameter), 289
- intpnt_nl_tol_mu_red (parameter), 289
- intpnt_nl_tol_near_rel (parameter), 289
- intpnt_nl_tol_pfeas (parameter), 290
- intpnt_nl_tol_rel_gap (parameter), 290
- intpnt_nl_tol_rel_step (parameter), 290
- intpnt_num_threads (parameter), 324
- intpnt_off_col_trh (parameter), 324
- intpnt_order_method (parameter), 324
- intpnt_regularization_use (parameter), 325
- intpnt_scaling (parameter), 325
- intpnt_solve_form (parameter), 325
- intpnt_starting_point (parameter), 326
- intpnt_tol_dfeas (parameter), 290
- intpnt_tol_dsafes (parameter), 291
- intpnt_tol_infeas (parameter), 291
- intpnt_tol_mu_red (parameter), 291
- intpnt_tol_path (parameter), 291
- intpnt_tol_pfeas (parameter), 292
- intpnt_tol_psafe (parameter), 292
- intpnt_tol_rel_gap (parameter), 292
- intpnt_tol_rel_step (parameter), 292
- intpnt_tol_step_size (parameter), 293
- mosek.iomode, 435
- isdoupparname (Task method), 218
- isintparname (Task method), 219
- isstrparname (Task method), 219
- itg
 - mosek.soltype.itg (example), 44
- itr_sol_file_name (parameter), 380
- mosek.language, 435
- lic_trh_expiry_wrn (parameter), 326
- license_allow_overuse (parameter), 326
- license_cache_time (parameter), 326
- license_check_time (parameter), 327
- license_debug (parameter), 327
- license_pause_time (parameter), 327
- license_suppress_expire_wrns (parameter), 328
- license_wait (parameter), 328
- mosek.liinfitem, 435
- linear dependency check, 94
- linear embedded network problem, 65
- Linear network flow problems, 61
- linear optimization, 18
- linear problem, 71
- linearity interval, 140
- linkfiletoenvstream (Env method), 175
- linkfiletotaskstream (Task method), 219
- log (parameter), 328
- log_bi (parameter), 329
- log_bi_freq (parameter), 329
- log_check_convexity (parameter), 329
- log_concurrent (parameter), 329
- log_cut_second_opt (parameter), 330
- log_factor (parameter), 330
- log_feasrepair (parameter), 330
- log_file (parameter), 331
- log_head (parameter), 331
- log_infeas_ana (parameter), 331
- log_intpnt (parameter), 331
- log_mio (parameter), 332
- log_mio_freq (parameter), 332
- log_nonconvex (parameter), 332
- log_optimizer (parameter), 332
- log_order (parameter), 333
- log_param (parameter), 333
- log_presolve (parameter), 333
- log_response (parameter), 333
- log_sensitivity (parameter), 334
- log_sensitivity_opt (parameter), 334

- log_sim (parameter), 334
- log_sim_freq (parameter), 335
- log_sim_minor (parameter), 335
- log_sim_network_freq (parameter), 335
- log_storage (parameter), 335
- lower_obj_cut (parameter), 293
- lower_obj_cut_finite_trh (parameter), 293
- LP format, 469
- lp_write_ignore_incompatible_items (parameter), 293
- 336
- makesolutionstatusunknown (Task method), 219
- mosek.mark, 436
- matrix format
 - column ordered, 56
 - row ordered, 56
 - triplets, 56
- max_num_warnings (parameter), 336
- mio.branch_dir (parameter), 336
- mio.branch_priorities_use (parameter), 336
- mio.construct_sol (parameter), 337
- mio.cont_sol (parameter), 337
- mio.cut_level_root (parameter), 337
- mio.cut_level_tree (parameter), 338
- mio.disable_term_time (parameter), 293
- mio.feaspump_level (parameter), 338
- mio.heuristic_level (parameter), 339
- mio.heuristic_time (parameter), 294
- mio.hotstart (parameter), 339
- mio.keep_basis (parameter), 339
- mio.local_branch_number (parameter), 339
- mio.max_num_branches (parameter), 340
- mio.max_num_relaxs (parameter), 340
- mio.max_num_solutions (parameter), 340
- mio.max_time (parameter), 294
- mio.max_time_aprx_opt (parameter), 295
- mio.mode (parameter), 341
- mio.near_tol_abs_gap (parameter), 295
- mio.near_tol_rel_gap (parameter), 295
- mio.node_optimizer (parameter), 341
- mio.node_selection (parameter), 342
- mio.optimizer_mode (parameter), 342
- mio.presolve_aggregate (parameter), 342
- mio.presolve_probing (parameter), 343
- mio.presolve_use (parameter), 343
- mio.rel_add_cut_limited (parameter), 296
- mio.rel_gap_const (parameter), 296
- mio.root_optimizer (parameter), 343
- mio.strong_branch (parameter), 344
- mio.tol_abs_gap (parameter), 296
- mio.tol_abs_relax_int (parameter), 296
- mio.tol_feas (parameter), 297
- mio.tol_rel_gap (parameter), 297
- mio.tol_rel_relax_int (parameter), 297
- mio.tol_x (parameter), 297
- mosek.miocontsoltype, 437
- mosek.miomode, 437
- mosek.mionodeseltype, 437
- mixed integer optimization, 40
- mixed-integer optimization, 113
- modelling
 - absolute value, 88
 - in cones, 78
 - market impact term, 90
 - Markowitz portfolio optimization, 89
 - minimizing a sum of norms, 80
 - portfolio optimization, 88
 - transaction costs, 90
- MPS format, 457
 - BOUNDS, 464
 - COLUMNS, 460
 - free, 468
 - NAME, 459
 - OBJNAME, 460
 - OBJSENSE, 459
 - QSECTION, 462
 - RANGES, 462
 - RHS, 461
 - ROWS, 460
- mosek.mpsformat, 438
- mosek.msgkey, 438
- netextraction
 - Task.netextraction (example), 67
- netextraction (Task method), 219
- netoptimize
 - Task.netoptimize (example), 63, 67
- netoptimize (Task method), 221
- Network flow problems
 - embedded, 103
 - optimizing, 103
- mosek.networkdetect, 438

- `nonconvex_max_iterations` (parameter), 344
- `nonconvex_tol_feas` (parameter), 298
- `nonconvex_tol_opt` (parameter), 298
- objective
 - defining, 23
 - linear, 23
 - quadratic, 75
 - vector, 71
- `objective_sense` (parameter), 344
- `mosek.objsense`, 438
- `mosek.onoffkey`, 439
- OPF format, 477
- OPF, writing, 16
- opf-write-hints
 - `mosek.iparam.opf_write_hints` (example), 16
- opf-write-parameters
 - `mosek.iparam.opf_write_parameters` (example), 16
- opf-write-problem
 - `mosek.iparam.opf_write_problem` (example), 16
- opf-write-solutions
 - `mosek.iparam.opf_write_solutions` (example), 16
- `opf_max_terms_per_line` (parameter), 344
- `opf_write_header` (parameter), 345
- `opf_write_hints` (parameter), 345
- `opf_write_parameters` (parameter), 345
- `opf_write_problem` (parameter), 346
- `opf_write_sol_bas` (parameter), 346
- `opf_write_sol_itg` (parameter), 346
- `opf_write_sol_itr` (parameter), 346
- `opf_write_solutions` (parameter), 347
- optimal solution, 73
- optimization
 - conic, 77
 - integer, 40, 113
 - mixed integer, 40
 - mixed-integer, 113
- optimize
 - `Task.optimize` (example), 16, 146
- optimize-penalty
 - `mosek.feasrepairtype.optimize_penalty` (example), 136
- optimizeconcurrent
 - `Task.optimizeconcurrent` (example), 109
- `optimizeconcurrent` (Task method), 222
- optimizer
 - `mosek.iparam.optimizer` (example), 107
- `optimizer` (parameter), 347
- `optimizer_max_time` (parameter), 298
- optimizers
 - concurrent, 106
 - conic interior-point, 104
 - convex interior-point, 104
 - linear interior-point, 96
 - parallel, 106
 - simplex, 101
- `optimizersummary` (Task method), 223
- `mosek.optimizertype`, 439
- `optimizetrm` (Task method), 223
- Optimizing
 - network flow problems, 103
- ORD format, 491
- `mosek.orderingtype`, 439
- parallel extensions, 105
- parallel interior-point, 96
- parallel optimizers
 - interior point, 96
- parallel solution, 105
- `param_comment_sign` (parameter), 380
- `param_read_case_name` (parameter), 347
- `param_read_file_name` (parameter), 380
- `param_read_ign_error` (parameter), 348
- `param_write_file_name` (parameter), 381
- `mosek.parametertype`, 440
- presolve, 93
 - eliminator, 94
 - linear dependency check, 94
- `presolve_elim_fill` (parameter), 348
- `presolve_eliminator_max_num_tries` (parameter), 348
- `presolve_eliminator_use` (parameter), 348
- `presolve_level` (parameter), 349
- `presolve_lindep_use` (parameter), 349
- `presolve_lindep_work_lim` (parameter), 349
- `presolve_tol_aij` (parameter), 298
- `presolve_tol_lin_dep` (parameter), 299
- `presolve_tol_s` (parameter), 299
- `presolve_tol_x` (parameter), 299
- `presolve_use` (parameter), 350
- `mosek.presolvemode`, 440

- primal feasible, 72
- primal certificate, 73
- primal infeasible, 72, 73
- primal-dual solution, 72
- primalsensitivity
 - Task.primalsensitivity (example), 146
- primalsensitivity (Task method), 223
- prntdata (Task method), 225
- prntparam (Task method), 226
- problem element
 - bounds
 - constraint, 19
 - variable, 19
 - constraint
 - bounds, 19
 - constraint matrix, 19
 - objective, linear, 19
 - variable
 - bounds, 19
 - variable vector, 18
- mosek.problemitem, 440
- mosek.problemtyp, 441
- mosek.prosta, 441
- putaij (Task method), 226
- putaijlist (Task method), 226
- putavec
 - Task.putavec (example), 23, 25, 29, 33, 37, 41
- putavec (Task method), 227
- putaveclist64 (Task method), 228
- putbound (Task method), 228
- putboundlist (Task method), 229
- putboundslice (Task method), 229
- putcfix (Task method), 230
- putcj (Task method), 230
- putclist (Task method), 230
- putcone (Task method), 231
- putcpdefaults (Env method), 175
- putdllpath (Env method), 175
- putdoupparam (Task method), 231
- putintparam
 - Task.putintparam (example), 107
- putintparam (Task method), 231
- putkeepdlls (Env method), 176
- putlicensedefaults (Env method), 176
- putmaxnumanz64 (Task method), 232
- putmaxnumcon (Task method), 232
- putmaxnumcone (Task method), 232
- putmaxnumqnz64 (Task method), 232
- putmaxnumvar (Task method), 233
- putnadoupparam (Task method), 233
- putnaintparam (Task method), 233
- putname (Task method), 233
- putnastrparam (Task method), 234
- putobjname (Task method), 234
- putobjsense (Task method), 234
- putparam (Task method), 234
- putqcon (Task method), 235
- putqconk
 - Task.putqconk (example), 33
- putqconk (Task method), 235
- putqobj
 - Task.putqobj (example), 29
- putqobj (Task method), 236
- putqobjij (Task method), 237
- putsolution (Task method), 237
- putsolutioni (Task method), 238
- putsolutionyi (Task method), 239
- putstrparam (Task method), 239
- puttaskname (Task method), 239
- putvarbranchorder (Task method), 240
- putvartype (Task method), 240
- putvartypelist
 - Task.putvartypelist (example), 41, 44
- putvartypelist (Task method), 240
- Python on Linux, 12
- Python script
 - executing (Linux), 13
- Python on Windows, 11
- qcqo_reformulate_rel_drop_tol (parameter), 299
- qo_separable_reformulation (parameter), 350
- mosek.qreadtype, 442
- quad
 - mosek.conetype.quad (example), 37
- quadratic constraint, 75
- quadratic constraints, example, 33
- quadratic objective, 75
- quadratic objective, example, 29
- quadratic optimization, 28, 75
- quadratic problem, 28
- read_add_anz (parameter), 350
- read_add_con (parameter), 350

- `read_add_cone` (parameter), 351
- `read_add_qnz` (parameter), 351
- `read_add_var` (parameter), 351
- `read_anz` (parameter), 351
- `read_con` (parameter), 352
- `read_cone` (parameter), 352
- `read_data_compressed` (parameter), 352
- `read_data_format` (parameter), 352
- `read_keep_free_con` (parameter), 353
- `read_lp_drop_new_vars_in_bou` (parameter), 353
- `read_lp_quoted_names` (parameter), 353
- `read_mps_bou_name` (parameter), 381
- `read_mps_format` (parameter), 354
- `read_mps_keep_int` (parameter), 354
- `read_mps_obj_name` (parameter), 381
- `read_mps_obj_sense` (parameter), 354
- `read_mps_quoted_names` (parameter), 355
- `read_mps_ran_name` (parameter), 381
- `read_mps_relax` (parameter), 355
- `read_mps_rhs_name` (parameter), 382
- `read_mps_width` (parameter), 355
- `read_q_mode` (parameter), 355
- `read_qnz` (parameter), 356
- `read_task_ignore_param` (parameter), 356
- `read_var` (parameter), 356
- `readbranchpriorities` (Task method), 241
- `readdata`
 - Task.readdata (example), 16, 136
- `readdata` (Task method), 241
- `readparamfile` (Task method), 241
- `readsolution` (Task method), 241
- `readsummary` (Task method), 242
- relaxation, continuous, 113
- relaxprimal
 - Task.relaxprimal (example), 136
- `relaxprimal` (Task method), 242
- `remove` (Task method), 244
- `removecone` (Task method), 244
- `mosek.rescodetype`, 442
- `resizetask` (Task method), 245
- `rquad`
 - `mosek.conetype.rquad` (example), 37
- scaling, 95
- `mosek.scalingmethod`, 443
- `mosek.scalingtype`, 443
- sensitivity analysis, 139
 - basis type, 141
 - optimal partition type, 142
- `sensitivity_all` (parameter), 357
- `sensitivity_file_name` (parameter), 382
- `sensitivity_optimizer` (parameter), 357
- `sensitivity_res_file_name` (parameter), 382
- `sensitivity_type` (parameter), 358
- `sensitivityreport` (Task method), 245
- `mosek.sensitivitytype`, 443
- `set_Stream` (Env method), 176
- `set_Stream` (Task method), 245
- `setdefaults` (Task method), 246
- shadow price, 140
- `sim_basis_factor_use` (parameter), 358
- `sim_degen` (parameter), 358
- `sim_dual_crash` (parameter), 359
- `sim_dual_phaseone_method` (parameter), 359
- `sim_dual_restrict_selection` (parameter), 359
- `sim_dual_selection` (parameter), 359
- `sim_exploit_dupvec` (parameter), 360
- `sim_hotstart` (parameter), 360
- `sim_hotstart_lu` (parameter), 361
- `sim_integer` (parameter), 361
- `sim_lu_tol_rel_piv` (parameter), 300
- `sim_max_iterations` (parameter), 361
- `sim_max_num_setbacks` (parameter), 361
- `sim_network_detect` (parameter), 362
- `sim_network_detect_hotstart` (parameter), 362
- `sim_network_detect_method` (parameter), 362
- `sim_non_singular` (parameter), 363
- `sim_primal_crash` (parameter), 363
- `sim_primal_phaseone_method` (parameter), 363
- `sim_primal_restrict_selection` (parameter), 363
- `sim_primal_selection` (parameter), 364
- `sim_refactor_freq` (parameter), 364
- `sim_reformulation` (parameter), 365
- `sim_save_lu` (parameter), 365
- `sim_scaling` (parameter), 365
- `sim_scaling_method` (parameter), 366
- `sim_solve_form` (parameter), 366
- `sim_stability_priority` (parameter), 366
- `sim_switch_optimizer` (parameter), 366
- `mosek.simdegen`, 443
- `mosek.simdupvec`, 444
- `mosek.simhotstart`, 444

- simplex optimizer, 101
- simplex_abs_tol_piv (parameter), 300
- `mosek.simreform`, 444
- `mosek.simseltype`, 444
- `sol_filter_keep_basic` (parameter), 367
- `sol_filter_keep_ranged` (parameter), 367
- `sol_filter_xc_low` (parameter), 382
- `sol_filter_xc_upr` (parameter), 383
- `sol_filter_xx_low` (parameter), 383
- `sol_filter_xx_upr` (parameter), 383
- `sol_quoted_names` (parameter), 367
- `sol_read_name_width` (parameter), 368
- `sol_read_width` (parameter), 368
- `mosek.solitem`, 445
- `mosek.solsta`, 445
- `mosek.soltype`, 446
- solution, optimal, 73
- solution, primal-dual, 72
- `solution_callback` (parameter), 368
- solutiondef
 - `Task.solutiondef` (example), 44
- `solutiondef` (Task method), 246
- solutionsummary
 - `Task.solutionsummary` (example), 16, 109
- `solutionsummary` (Task method), 246
- `mosek.solveform`, 447
- `solvewithbasis` (Task method), 246
- sparse vector, 56
- `mosek.stakey`, 447
- `mosek.startpointtype`, 447
- `stat_file_name` (parameter), 384
- `stat_key` (parameter), 384
- `stat_name` (parameter), 384
- stream
 - attaching, 23
- `mosek.streamtype`, 448
- `strtoconetype` (Task method), 247
- `strtosk` (Task method), 247
- Task
 - constructors, 178
- task, creatig, 16
- `Task.Task` (constructor), 178
- thread safety, 155
- `timing_level` (parameter), 369
- `undefsolution` (Task method), 248
- `upper_obj_cut` (parameter), 300
- `upper_obj_cut_finite_trh` (parameter), 300
- `mosek.value`, 448
- variables
 - decision, 71, 457
 - lower limit, 72, 457
 - number of, 18
 - upper limit, 72, 457
- `mosek.variabletype`, 448
- vector format
 - full, 55
 - sparse, 56
- Warning
 - constructors, 249
- `Warning.Warning` (constructor), 249
- `warning_level` (parameter), 369
- write-data-format
 - `mosek.iparam.write_data_format` (example), 16
- `write_bas_constraints` (parameter), 369
- `write_bas_head` (parameter), 369
- `write_bas_variables` (parameter), 370
- `write_data_compressed` (parameter), 370
- `write_data_format` (parameter), 370
- `write_data_param` (parameter), 371
- `write_free_con` (parameter), 371
- `write_generic_names` (parameter), 371
- `write_generic_names_io` (parameter), 371
- `write_int_constraints` (parameter), 372
- `write_int_head` (parameter), 372
- `write_int_variables` (parameter), 372
- `write_lp_gen_var_name` (parameter), 384
- `write_lp_line_width` (parameter), 373
- `write_lp_quoted_names` (parameter), 373
- `write_lp_strict_format` (parameter), 373
- `write_lp_terms_per_line` (parameter), 373
- `write_mps_int` (parameter), 374
- `write_mps_obj_sense` (parameter), 374
- `write_mps_quoted_names` (parameter), 374
- `write_mps_strict` (parameter), 374
- `write_precision` (parameter), 375
- `write_sol_constraints` (parameter), 375
- `write_sol_head` (parameter), 375
- `write_sol_variables` (parameter), 376
- `write_task_inc_sol` (parameter), 376
- `write_xml_mode` (parameter), 376

`writebranchpriorities` (Task method), 248

`writedata`

 Task.writedata (example), 16

`writedata` (Task method), 248

`writeparamfile` (Task method), 249

`writesolution` (Task method), 249

xml format, 489

`mosek.xmlwriteroutputtype`, 448