

**The MOSEK Python API manual.
Version 5.0 (Revision 138).**



www.mosek.com

Published by MOSEK ApS, Denmark.

Copyright 1999-2009 MOSEK ApS, Denmark

Disclaimer: MOSEK ApS (the author of MOSEK) accepts no responsibility for damages resulting from the use of the MOSEK software and makes no warranty, either expressed or 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	File formats	3
1.2	Optimizers	3
1.3	API changes	4
1.4	License system	4
1.5	Other changes	4
1.6	Interfaces	4
1.7	Supported platforms	4
2	About this manual	5
3	Getting support and help	7
3.1	MOSEK documentation	7
3.2	Additional reading	7
4	Testing installation and Running examples	9
4.1	Microsoft Windows Platform	9
4.1.1	Running an example	10
4.2	Linux Platform	10
4.2.1	Executing a Python program	11
5	Basic API tutorial	13
5.1	The basics	13
5.1.1	The environment and the task	13
5.1.2	A simple working example	14

5.1.3	Compiling and running examples	16
5.2	Linear optimization	16
5.2.1	Example: lo1	17
5.2.2	An alternative implementation: lo2	22
5.3	Quadratic optimization	25
5.3.1	Example: Quadratic objective	26
5.3.2	Example: Quadratic constraints	29
5.4	Conic optimization	32
5.4.1	Example: cqo1	33
5.5	Integer optimization	35
5.5.1	Example: milo1	36
5.5.2	Specifying an initial solution	38
5.5.3	Example: Specifying an integer solution	38
5.6	Problem modification and reoptimization	41
5.6.1	A production planning problem	41
5.6.2	Changing the A matrix	44
5.6.3	Appending variables	44
5.6.4	Reoptimization	45
5.6.5	Appending constraints	45
5.7	Efficiency considerations	46
5.7.1	API overhead	47
5.8	Conventions employed in the API	47
5.8.1	Naming conventions for arguments	47
5.8.2	Vector formats	50
5.8.3	Matrix formats	50
5.8.4	Array objects	52
5.8.5	Typical problems using the Python API	53
6	Modelling	55
6.1	Linear optimization	55
6.1.1	Duality for linear optimization	56
6.1.2	Primal and dual infeasible case	58
6.2	Linear network flow problems	59

6.3	Quadratic and quadratically constrained optimization	59
6.3.1	A general recommendation	59
6.3.2	Reformulating as a separable quadratic problem	60
6.4	Conic optimization	61
6.4.1	Duality for conic optimization	62
6.4.2	The dual of the dual	63
6.4.3	Infeasibility	63
6.4.4	Examples	63
6.4.5	Potential pitfalls in conic optimization	67
6.5	Recommendations	69
6.5.1	Avoid nearly infeasible models	70
6.6	Examples continued	70
6.6.1	The absolute value	70
6.6.2	The Markowitz portfolio model	71
7	The optimizers for continuous problems	75
7.1	How an optimizer works	75
7.1.1	Presolve	75
7.1.2	Dualizer	77
7.1.3	Scaling	77
7.1.4	Using multiple CPU's	77
7.2	Linear optimization	78
7.2.1	Optimizer selection	78
7.2.2	The interior-point optimizer	78
7.2.3	The simplex based optimizer	79
7.2.4	The interior-point or the simplex optimizer?	80
7.2.5	The primal or the dual simplex variant?	81
7.3	Linear network optimization	81
7.3.1	Network flow problems	81
7.3.2	Embedded network problems	81
7.4	Conic optimization	82
7.4.1	The interior-point optimizer	82
7.5	Nonlinear convex optimization	82

7.5.1	The interior-point optimizer	82
7.6	Solving problems in parallel	83
7.6.1	Thread safety	83
7.6.2	The parallelized interior-point optimizer	83
7.6.3	The concurrent optimizer	83
7.6.4	A more flexible concurrent optimizer	86
8	The optimizer for mixed integer problems	89
8.1	Some notation	89
8.2	An important fact about integer optimization problems	90
8.3	How the integer optimizer works	90
8.3.1	Presolve	91
8.3.2	Heuristic	91
8.3.3	The optimization phase	91
8.4	Termination criterion	91
8.5	How to speed up the solution process	92
9	Analyzing infeasible problems	95
9.1	Example: Primal infeasibility	95
9.1.1	Locating the cause of primal infeasibility	97
9.1.2	Locating the cause of dual infeasibility	97
9.1.3	The infeasibility report	98
9.2	Theory concerning infeasible problems	102
9.2.1	Certificat of primal infeasibility	102
9.2.2	Certificat of dual infeasibility	103
10	Primal feasibility repair	105
10.1	The main idea	105
10.2	Feasibility repair in MOSEK	107
10.2.1	Usage of negative weights	107
10.2.2	Automatical naming	107
10.2.3	Feasibility repair using the API	108
10.2.4	An example	108

11 Sensitivity analysis	111
11.1 Introduction	111
11.2 Restrictions	111
11.3 References	111
11.4 Sensitivity analysis for linear problems	112
11.4.1 The optimal objective value function	112
11.4.2 The basis type sensitivity analysis	113
11.4.3 The optimal partition type sensitivity analysis	114
11.4.4 Example: Sensitivity analysis	115
11.5 Sensitivity analysis from the MOSEK API	118
11.6 Sensitivity analysis with the command line tool	121
11.6.1 Sensitivity analysis specification file	122
11.6.2 Example: Sensitivity analysis from command line	123
11.6.3 Controlling log output	124
12 API developer guidelines	125
12.1 Turn on logging	125
12.2 Turn on data checking	126
12.3 Debugging an optimization task	126
12.4 Error handling	126
12.5 Check the problem status and solution status	126
12.6 Important API limitations	126
12.6.1 Thread safety	126
12.7 Bug reporting	127
13 API reference	129
13.1 API Functionality	129
13.1.1 Reading and writing data files.	129
13.1.2 Solutions.	130
13.1.3 Memory allocation and deallocation.	131
13.1.4 Changing problem specification.	131
13.1.5 Delete problem elements (variables,constraints,cones).	133
13.1.6 Add problem elements (variables,constraints,cones).	133

13.1.7 Inspect problem specification.	133
13.1.8 Conic constraints.	135
13.1.9 Bounds.	135
13.1.10 Output stream functions.	136
13.1.11 Objective function.	136
13.1.12 Inspect statistics from the optimizer.	137
13.1.13 Parameters (set/get).	138
13.1.14 Naming.	139
13.1.15 Preallocating space for problem data.	139
13.1.16 Integer variables.	140
13.1.17 Quadratic terms.	141
13.1.18 Diagnosing infeasibility.	141
13.1.19 Optimization.	141
13.1.20 Sensitivity analysis.	142
13.1.21 Testing data validity.	142
13.1.22 Solving with the basis.	142
13.1.23 Initialization of environment.	142
13.1.24 Change A.	143
13.2 Class <code>mosek.ArrayLengthException</code>	143
13.3 Class <code>mosek.Env</code>	143
13.3.1 Constructors	143
13.3.2 Attributes	144
13.3.3 Methods	144
13.4 Class <code>mosek.Error</code>	148
13.4.1 Constructors	148
13.5 Class <code>mosek.Exception</code>	149
13.5.1 Constructors	149
13.6 Class <code>mosek.Task</code>	149
13.6.1 Constructors	149
13.6.2 Attributes	150
13.6.3 Methods	150
13.7 Class <code>mosek.Warning</code>	217

13.7.1 Constructors	217
14 Parameter reference	219
14.1 Parameter groups	219
14.1.1 Logging parameters.	219
14.1.2 Basis identification parameters.	221
14.1.3 The Interior-point method parameters.	221
14.1.4 Simplex optimizer parameters.	224
14.1.5 Primal simplex optimizer parameters.	225
14.1.6 Dual simplex optimizer parameters.	225
14.1.7 Network simplex optimizer parameters.	225
14.1.8 Nonlinear convex method parameters.	226
14.1.9 The conic interior-point method parameters.	226
14.1.10 The mixed integer optimization parameters.	227
14.1.11 Presolve parameters.	229
14.1.12 Termination criterion parameters.	229
14.1.13 Progress call-back parameters.	231
14.1.14 Non-convex solver parameters.	232
14.1.15 Feasibility repair parameters.	232
14.1.16 Optimization system parameters.	232
14.1.17 Output information parameters.	233
14.1.18 Extra information about the optimization problem.	234
14.1.19 Overall solver parameters.	235
14.1.20 Behavior of the optimization task.	236
14.1.21 Data input/output parameters.	237
14.1.22 Solution input/output parameters.	242
14.1.23 Infeasibility report parameters.	243
14.1.24 License manager parameters.	244
14.1.25 Data check parameters.	244
14.2 Double parameters	245
14.3 Integer parameters	265
14.4 String parameter types	334

15 Response codes	343
16 Constants	423
16.1 Constraint or variable access modes	426
16.2 Basis identification	426
16.3 Bound keys	426
16.4 Specifies the branching direction.	427
16.5 Progress call-back codes	427
16.6 Types of convexity checks.	432
16.7 Compression types	432
16.8 Cone types	433
16.9 CPU type	433
16.10Data format types	434
16.11Double information items	434
16.12Double values	438
16.13Feasibility repair types	438
16.14Integer information items.	438
16.15Information item types	443
16.16Input/output modes	443
16.17Bound keys	443
16.18Continuous mixed integer solution type	443
16.19Integer restrictions	444
16.20Mixed integer node selection types	444
16.21MPS file format type	444
16.22Message keys	445
16.23Network detection method	445
16.24Objective sense types	445
16.25On/off	445
16.26Optimizer types	445
16.27Ordering strategies	446
16.28Parameter type	447
16.29Presolve method.	447
16.30Problem data items	447

16.31	Problem types	447
16.32	Problem status keys	448
16.33	Interpretation of quadratic terms in MPS files	449
16.34	Response code type	449
16.35	Scaling type	449
16.36	Sensitivity types	450
16.37	Degeneracy strategies	450
16.38	Hot-start type employed by the simplex optimizer	450
16.39	Simplex selection strategy	450
16.40	Solution items	451
16.41	Solution status keys	451
16.42	Solution types	452
16.43	Solve primal or dual form	453
16.44	Status keys	453
16.45	Starting point types	453
16.46	Stream types	454
16.47	Integer values	454
16.48	Variable types	454
16.49	XML writer output mode	454
A	The MPS file format	455
A.1	The MPS file format	455
A.1.1	An example	457
A.1.2	NAME	457
A.1.3	OBJSENSE (optional)	457
A.1.4	OBJNAME (optional)	458
A.1.5	ROWS	458
A.1.6	COLUMNS	458
A.1.7	RHS (optional)	459
A.1.8	RANGES (optional)	460
A.1.9	QSECTION (optional)	460
A.1.10	BOUNDS (optional)	462
A.1.11	CSECTION (optional)	463

A.1.12 ENDATA	465
A.2 Integer variables	465
A.3 General limitations	466
A.4 Interpretation of the MPS format	466
A.5 The free MPS format	466
B The LP file format	467
B.1 A warning	467
B.2 The LP file format	467
B.2.1 The sections	468
B.2.2 LP format peculiarities	471
B.2.3 The strict LP format	473
B.2.4 Formatting of an LP file	473
C The OPF format	475
C.1 Intended use	475
C.2 The file format	475
C.2.1 Sections	476
C.2.2 Numbers	480
C.2.3 Names	480
C.3 Parameters section	481
C.4 Writing OPF files from MOSEK	481
C.5 Examples	482
C.5.1 Linear example <code>lo1.opf</code>	482
C.5.2 Quadratic example <code>qo1.opf</code>	483
C.5.3 Conic quadratic example <code>cqo1.opf</code>	483
C.5.4 Mixed integer example <code>mi1o1.opf</code>	484
D The XML (OSiL) format	487
E The ORD file format	489
E.1 An example	489
F The solution file format	491

F.1 The basic and interior solution files	491
F.2 The integer solution file	492

License agreement

Before using the MOSEK software, please read the license agreement available in the distribution `incd`
`mosek\5\license\index.html`

Chapter 1

Changes and new features in MOSEK

The section presents improvements and new features added to MOSEK in version 5.0.

1.1 File formats

- The OSiL XML format for linear problems is now supported as output-only format.
- The new Optimization Problem file Format (OPF) is now available. It incorporates linear, quadratic, and conic problems in a single format, as well as parameter settings and solutions.
- The OBJNAME section is now supported in the MPS format.

1.2 Optimizers

- The interior-point solver is about 20% faster on average for large linear problems, compared to MOSEK 4.0.
- The dual simplex solver is about 40% faster on average compared to MOSEK 4.0.
- For the primal simplex solver, handling of problems with long slim structure has been improved.
- For both simplex optimizers numerical stability, hot-start efficiency and degeneracy handling has been improved substantially.
- A simplex network flow optimizer is now available. In many cases the specialized simplex optimizer can solve a pure network flow optimization problem up to 10 times faster than the standard simplex optimizer.
- Presolve is now by default turned on for hot-start with the simplex optimizers.

- The mixed integer optimizer now includes the feasibility pump heuristic to find a good initial feasible solution.
- Full support for setting branching priorities on integer constrained variables.

1.3 API changes

- The function `Task.putobjsense` has been introduced. This should be used to define objective sense instead of the parameter `mosek.iparam.objective_sense`.

1.4 License system

- The Flexlm license software has been upgraded to version 11.4.
- Dongles are supported in 64 bit Windows.

1.5 Other changes

- The documentation has been improved. Each interface now have a complete dedicated manual, and many code examples have been added. The HTML version has been subject to heavy cosmetical changes.

1.6 Interfaces

- A complete Python interface is now available.
- The MATLAB interface supports the MATLAB versions R2006a, R2006b, and R2007a.
- The general convex interface has been disabled in the Java and .NET interfaces.
- The Java API provides an interface to the native `scopt` functionality.

1.7 Supported platforms

- Mac OSX 32 bit for x86 version has been added.
- Solaris 32 bit for x86 version has been added
- Solaris 64 bit for x86 version has been added.

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 12 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 6 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 6 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 10 which deals with analysis of infeasible problems,
- Chapter 11 about the sensitivity analysis interface, and

Finally, the Python API reference material is located in

- Chapter 13 which lists all types and functions,
- Chapter 14 which lists all available parameters,
- Chapter 15 which lists all response codes, and
- Chapter 16 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\5\help\index.html`

in the distribution.

3.2 Additional reading

In this manual it is assumed the reader is familiar with mathematics and in particular mathematical optimization. Some introduction to linear programming can be found in books such as “Linear programming” by Chvátal [13] or “Computer Solution of Linear Programs” by Nazareth [18]. For more theoretical aspects see for example “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 modelling 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 requires a MOSEK installation, a full Python installation and the Numerical Python extension. A Python installer can be obtained from the official site

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

or packaged from

<http://www.activestate.com/Products/ActivePython/>

The Numeric extension providing arrays and mathematical functionality can be obtained from

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

This package is not required, but the MOSEK Python API can use Numeric arrays more efficiently than generic arrays or lists. The Numeric package is expected to be replaced by NumPy in the near future.

4.1 Microsoft Windows Platform

To use the MOSEK Python interface, a working MOSEK installation must be present — see the MOSEK Installation manual for instructions. In the following we assume that MOSEK is installed in the default directory

`C:\Program Files\mosek`

Please note that this may be different for other language settings than US English. The MOSEK Python interface is defined in

```
C:\Program Files\mosek\5\tools\platform\win\bin\pymosek.pyd
```

or

```
C:\Program Files\mosek\5\tools\platform\win64\bin\pymosek.pyd
```

depending on the platform.

The distributed Python examples are found in

```
C:\Program Files\mosek\5\tools\examples\python
```

4.1.1 Running an example

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

```
C:  
cd "\Program Files\mosek\5\tools\examples\python"
```

then to execute example `lo1` type

```
python lo1.py
```

4.2 Linux Platform

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

The MOSEK Python API is defined in the library

```
pymosek.so
```

in one of the directories

```
~/mosek/5/tools/platform/linux32x86/bin/pymosek.so  
~/mosek/5/tools/platform/linux64x86/bin/pymosek.so
```

depending on the platform. All other required libraries reside in the same directory as this module.

The distributed Python examples are found in

```
C:/mosek/5/tools/examples/python
```

Before the MOSEK Python 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.so`.

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 `pymosek.so`. 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=$LD_LIBRARY_PATH:"$HOME/mosek/5/tools/platform/<arch>/bin"
export PYTHONPATH=$PYTHONPATH:"$HOME/mosek/5/tools/platform/<arch>/bin"
```

where `<arch>` is either “linux32x86” (on 32-bit Linux) or “linux64x86” (on 64-bit linux). 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/5/tools/platform/<arch>/bin"
else
  export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:"$HOME/mosek/5/tools/platform/<arch>/bin"
fi

if [ -z "$PYTHONPATH" ]; then
  export PYTHONPATH="$HOME/mosek/5/tools/platform/<arch>/bin"
else
  export PYTHONPATH=$PYTHONPATH:"$HOME/mosek/5/tools/platform/<arch>/bin"
fi
```

where `<arch>` is either “linux32x86” (on 32-bit systems), “linux64x86” (on AMD64 or EM64T) or “linuxia64” (on Itanium). Note that other shells may require a different syntax.

4.2.1 Executing a Python program

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.

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 (`pymosek.Env`) object.
3. Set up some environment specific data and initialize the environment object.
4. Create a task (`pymosek.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 (`pymosek.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 (`pymosek.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 pymosek
import Numeric

# Create a Mosek handle
msk = pymosek.mosek()
# Create an environment
env = msk.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.

```
#
# Copyright: Copyright (c) 1998-2007 MOSEK ApS, Denmark. All rights reserved.
#
# File:      simple.py
#
# Purpose: Demonstrates a very simple example using MOSEK by
```

```

# reading a problem file, solving the problem and
# writing the solution to a file.
#

import sys
import pymosek as pymsk

if len(sys.argv) <= 1:
    print "Missing argument. The syntax is:"
    print " simple inputfile [ solutionfile ]"
else:
    # Make mosek handle
    msk = pymsk.mosek ()

    # Make mosek environment.
    env = pymsk.Env ()

    # Initialize the environment.
    env.init ()

    # Create a task object linked with the environment env.
    # We create it initially with 0 variables and 0 columns,
    # since we don't know the size of the problem.
    task = env.Task (0,0)

    # We assume that a problem file was given as the first command
    # line argument (received in 'args')
    task.readdata (sys.argv[1])

    # Solve the problem
    task.optimize()

    # Print a summary of the solution
    task.solutionsummary(pymsk.streamtype.log)

    # If an output file was specified, write a solution
    if len(sys.argv) > 2:
        # We define the output format to be OPF, and tell MOSEK to
        # leave out parameters and problem data from the output file.
        task.putintparam (pymsk.iparam.write_data_format,    pymsk.dataformat.op)
        task.putintparam (pymsk.iparam.opf_write_solutions,  pymsk.onoffkey.on)
        task.putintparam (pymsk.iparam.opf_write_hints,      pymsk.onoffkey.off)
        task.putintparam (pymsk.iparam.opf_write_parameters, pymsk.onoffkey.off)
        task.putintparam (pymsk.iparam.opf_write_problem,    pymsk.onoffkey.off)

        task.writedata(sys.argv[2])

```

5.1.2.1 Writing a problem to a file

Use the `Task.writedata` function to write a problem to a file. By default MOSEK will determine the output file format by the extension of the filename, for example to write an OPF file:

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

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 (`pymosek.Task`) provides a number of methods to operate on the task specific data, all of which are listed in Section 13.6.

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 14.

5.1.3 Compiling and running examples

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

`mosek/5/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 scalar 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 Example: lo1

The following is an example of a linear optimization problem:

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

having the bounds

$$\begin{array}{llll} 0 & \leq & x_0 & \leq & \infty, \\ 0 & \leq & x_1 & \leq & 10, \\ 0 & \leq & x_2 & \leq & \infty, \\ 0 & \leq & x_3 & \leq & \infty. \end{array} \quad (5.5)$$

5.2.1.1 Source code

The data structures used in the following example will be explained in detail in [5.8](#).

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

```
mosek\5\tools\examp\
```

```

##
#   Copyright: Copyright (c) 1998-2007 MOSEK ApS, Denmark. All rights reserved.
#
#   File:      lo1.py
#
#   Purpose: Demonstrates how to solve small linear
#             optimization problem using the MOSEK Python API.
#
##

import sys
import os

import pymosek as mosek

try:
    from Numeric import array, Float, zeros, ones
except ImportError:
    from mosekarr import array, Float, zeros, ones

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

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

# We might write everything directly as a script, but it looks nicer
# to create a function.
def main():
    # Open MOSEK and create an environment and task
    # Create a handle to MOSEK
    mskhandle = mosek.mosek()
    # Make a MOSEK environment
    env = mskhandle.Env()
    # Attach a printer to the environment
    env.set_Stream(mosek.streamtype.log, streamprinter)
    # Initialize the environment
    env.init()

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

    # Bound keys for constraints
    bkc = array([mosek.boundkey.fx,
                 mosek.boundkey.lo,
                 mosek.boundkey.up])
    # Bound values for constraints
    blc = array([30.0, 15.0, -inf])
    buc = array([30.0, +inf, 25.0])

```

```

# Bound keys for variables
bkc = array ([mosek.boundkey.lo,
              mosek.boundkey.ra,
              mosek.boundkey.lo,
              mosek.boundkey.lo])
# Bound values for variables
blx = array ([ 0.0,  0.0,  0.0,  0.0])
bux = array ([+inf, 10.0, +inf, +inf])

# Objective coefficients
c = array([ 3.0, 1.0, 5.0, 1.0 ])

# We input the A matrix column-wise
# asub contains row indexes
asub = array([ 0, 1,
              0, 1, 2,
              0, 1,
              1, 2])
# acof contains coefficients
acof = array([ 3.0, 2.0,
              1.0, 1.0, 2.0,
              2.0, 3.0,
              1.0, 3.0 ])
# aptrb and aptre contains the offsets into asub and acof where
# columns start and end respectively
aptrb = array([ 0, 2, 5, 7 ])
aptre = array([ 2, 5, 7, 9 ])

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

# Load data into task
print 'ptrb =',aptrb
print 'ptre =',aptre
print "numvar =",numvar
print "numcon =",numcon
task.inputdata (numcon,numvar,
               c, # linear objective coefficients
               0.0, # objective fixed value
               aptrb,
               aptre,
               asub,acof,
               bkc,blc,buc,
               bkc,blc,buc,
               bkc,blc,buc)
task.putobjsense(mosek.objsense.maximize)

task.optimize ()
print "Solution summary"

task.solutionssummary (mosek.streamtype.msg);

xx = zeros(numvar, Float)
print "point 13"
task.getsolutionslice(mosek.soltype.bas,
                     mosek.solitem.xx,
                     0, numvar,
                     xx)
print "point 14"

```

```

    print "x =", xx

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

```

5.2.1.2 Example code comments

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

```

# Create a handle to MOSEK
mskhandle = mosek.mosek ()
# Make a MOSEK environment
env = mskhandle.Env ()
# Attach a printer to the environment
env.set_Stream (mosek.streamtype.log, streamprinter)
# Initialize the environment
env.init ()

```

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.

MOSEK optimization task: Next, an empty task object is created:

```

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

```

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

Inputting the problem data: When the task has been created, data can be loaded into it. This happens here:

```

# Load data into task
print 'ptrb =', aptrb
print 'ptre =', aptre
print "numvar =", numvar
print "numcon =", numcon
task.inputdata (numcon, numvar,

```

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 5.1: Interpretation of the bound keys.

```

c, # linear objective coefficients
0.0, # objective fixed value
ptrb,
ptre,
asub,acof,
bkc,blc,buc,
bkx,blx,bux)

```

There are several different ways to set up an optimization problem; in this case we loaded the whole problem using a single function, `Task.inputdata`.

The `ptrb`, `ptre`, `asub`, and `aval` arguments define the constraint matrix A in the column ordered sparse format (for details, see Section 5.8.3.2).

The `c` argument is a full vector defining the objective function.

The precise relation between the arguments and the mathematical expressions in (5.1)–(5.3) is as follows.

- The linear terms in the constraints:

$$a_{\text{sub}[t],j} = \text{val}[t], \quad t = \text{ptrb}[j], \dots, \text{ptre}[j] - 1, \quad j = 0, \dots, \text{numvar} - 1. \quad (5.6)$$

For an illustrated example of the meaning of `ptrb` and `ptre` see Section 5.8.3.2.

- The linear terms in the objective:

$$c_j = c[j], \quad j = 0, \dots, \text{numvar} - 1 \quad (5.7)$$

- The bounds for the constraints are specified using the `bkc`, `blc`, and `buc` variables. The components of the `bkc` integer array specify the type of the bounds according to Table 5.1. For instance `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

$$l_k^c = \text{blc}[k], \quad k = 0, \dots, \text{numcon} - 1 \quad (5.8)$$

and

$$u_k^c = \text{buc}[k], \quad k = 0, \dots, \text{numcon} - 1. \quad (5.9)$$

- The bounds on the variables are specified using the `bkx`, `blx`, and `bux` variables. The components in the `bkx` integer array specifies the type of the bounds according to Table 5.1. The numerical values for the lower bounds on the variables are given by

$$l_j^x = \text{blx}[j], \quad j = 0, \dots, \text{numvar} - 1. \quad (5.10)$$

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

$$u_j^x = \text{bux}[j], \quad j = 0, \dots, \text{numvar} - 1. \quad (5.11)$$

Optimization: After set-up the task can be optimized.

```
task.optimize ()
```

Outputting the solution: Finally, the primal solution is retrieved and printed.

```
xx = zeros(numvar, Float)
print "point 13"
task.getsolutionslice(mosek.soltype.bas,
                      mosek.solitem.xx,
                      0, numvar,
                      xx)
print "point 14"
print "x =", xx
```

The `Task.getsolutionslice` function obtains a “slice” of the solution. In fact MOSEK may compute several solutions depending on the optimizer employed. In this example the *basic solution* is requested, specified by `mosek.soltype.bas`. The `mosek.solitem.xx` specifies that we want the variable values of the solution, and the following 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:

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

The types of exceptions that MOSEK can throw can be seen in 13.4 and 13.7.

5.2.2 An alternative implementation: `lo2`

In the previous example the problem data is loaded in one chunk. It is often more convenient to add one constraint or one variable at a time — this is possible using the following approach:

- Before a constraint or a variable can be used it has to be added with `Task.append` or a similar function. By default the appended constraints will be empty and the bounds of the appended constraints are infinite. Variables are fixed at zero.

- The objective function is specified using `Task.putcfix` and `Task.putcj`.
- The lower and upper bounds on the constraints and variables are specified using `Task.putbound`.
- The non-zero entries in A are added one column at a time using `Task.putavec`.

```
#
# Copyright: Copyright (c) 1998-2007 MOSEK ApS, Denmark. All rights reserved.
#
# File:      lo2.py
#
# Purpose: Demonstrates how to solve small linear
#          optimization problem using the MOSEK Python API.
##

import sys

import pymosek as mosek

from mosekarr import array, Float, zeros, ones

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

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

# We might write everything directly as a script, but it looks nicer
# to create a function.
def main():
    # Create a handle to MOSEK
    mskhandle = mosek.mosek()
    # Make a MOSEK environment
    env = mskhandle.Env()
    # Attach a printer to the environment
    env.set_Stream(mosek.streamtype.log, streamprinter)
    # Initialize the environment
    env.init()

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

    # Bound keys for constraints
    bkc = array([mosek.boundkey.fx,
                 mosek.boundkey.lo,
                 mosek.boundkey.up])
    # Bound values for constraints
    blc = array([30.0, 15.0, -inf])
    buc = array([30.0, +inf, 25.0])

    # Bound keys for variables
```

```

bkc = array ([mosek.boundkey.lo,
              mosek.boundkey.ra,
              mosek.boundkey.lo,
              mosek.boundkey.lo])
# Bound values for variables
blx = array ([ 0.0,  0.0,  0.0,  0.0])
bux = array ([+inf, 10.0, +inf, +inf])

# Objective coefficients
csub = array([ 0,  1,  2,  3 ])
cval = array([ 3.0, 1.0, 5.0, 1.0 ])

# We input the A matrix column-wise
# asub contains row indexes
asub = array([ 0, 1,
              0, 1, 2,
              0, 1,
              1, 2])
# acof contains coefficients
acof = array([ 3.0, 2.0,
              1.0, 1.0, 2.0,
              2.0, 3.0,
              1.0, 3.0 ])
# aptrb and aptre contains the offsets into asub and acof where
# columns start and end respectively
aptrb = array([ 0, 2, 5, 7 ])
aptre = array([ 2, 5, 7, 9 ])

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

# Append the constraints
task.append(mosek.acemode.con,numcon);

# Append the variables.
task.append(mosek.acemode.var,numvar);

# Input objective
task.putcfix(0.0)
task.putclist(csub,cval);

# Put constraint bounds
task.putboundslice(mosek.acemode.con,
                   0, numcon,
                   bkc, blc, buc);

# Put variable bounds
task.putboundslice(mosek.acemode.var,
                   0, numvar,
                   bkc, blx, bux);

# Input A non-zeros by columns
for j in range(numvar):
    ptrb,ptre = aptrb[j],aptre[j]
    task.putavec(mosek.acemode.var,j,
                 asub[ptrb:ptre],

```



```

                                acof[ptrb:ptrel])

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

# Optimize the task
task.optimize();

# Output a solution
xx = zeros(numvar, Float)
task.getsolutionslice(mosek.soltype.bas,
                      mosek.solitem.xx,
                      0,numvar,
                      xx)

print "xx =", xx

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

```

5.3 Quadratic optimization

It is possible to solve quadratic and quadratically constrained convex problems using MOSEK. This class of problems can be formulated as follows:

$$\begin{aligned}
 & \text{minimize} && \frac{1}{2}x^T Q^o x + c^T x + c^f \\
 & \text{subject to} && l_k^c \leq \frac{1}{2}x^T Q^k x + \sum_{j=0}^{n-1} a_{k,j} x_j \leq u_k^c, \quad k = 0, \dots, m-1, \\
 & && l^x \leq x \leq u^x, \quad j = 0, \dots, n-1.
 \end{aligned} \tag{5.12}$$

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

A very important restriction in MOSEK is that the problem should be convex. This implies that the matrix Q^o should be positive semi-definite and that the k th constraint must be of the form

$$l_k^c \leq \frac{1}{2}x^T Q^k x + \sum_{j=0}^{n-1} a_{k,j} x_j \tag{5.13}$$

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

$$\frac{1}{2}x^T Q^k x + \sum_{j=0}^{n-1} a_{k,j} x_j \leq u_k^c. \quad (5.14)$$

with a positive semi-definite Q^k . This implies that quadratic equalities are specifically *not* allowed.

5.3.1 Example: Quadratic objective

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

$$\begin{aligned} & \text{minimize} && x_1^2 + 0.1x_2^2 + x_3^2 - x_1x_3 - x_2 \\ & \text{subject to} && 1 \leq x_1 + x_2 + x_3 \\ & && x \geq 0 \end{aligned} \quad (5.15)$$

This can be written equivalently as

$$\begin{aligned} & \text{minimize} && 1/2x^T Q^o x + c^T x \\ & \text{subject to} && Ax \geq b \\ & && x \geq 0, \end{aligned} \quad (5.16)$$

where

$$Q^o = \begin{bmatrix} 2 & 0 & -1 \\ 0 & 0.2 & 0 \\ -1 & 0 & 2 \end{bmatrix}, \quad c = \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix}, \quad A = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}, \quad \text{and } b = 1. \quad (5.17)$$

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

```
##
#   Copyright: Copyright (c) 1998-2007 MOSEK ApS, Denmark. All rights reserved.
#
#   File:      qo1.py
#
#   Purpose: Demonstrate how to solve a quadratic
#             optimization problem using the MOSEK Python API.
##

import sys
import os

import pymosek as mosek

from mosekarr import array, Float, zeros, ones

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

```

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

# We might write everything directly as a script, but it looks nicer
# to create a function.
def main():
    # Open MOSEK and create an environment and task
    # Create a handle to MOSEK
    mskhandle = mosek.mosek()
    # Make a MOSEK environment
    env = mskhandle.Env()
    # Attach a printer to the environment
    env.set_Stream(mosek.streamtype.log, streamprinter)
    # Initialize the environment
    env.init()
    # Create a task
    task = env.Task()

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

    bkc = array([ mosek.boundkey.lo,
                  mosek.boundkey.lo,
                  mosek.boundkey.lo ])
    blx = array([ 0.0, 0.0, 0.0 ])
    bux = array([ inf, inf, inf ])

    c = array([ 0.0, -1.0, 0.0 ])

    asub = array([ 0, 0, 0 ])
    aval = array([ 1.0, 1.0, 1.0 ])
    aptrb = array([ 0, 1, 2 ])
    aptre = array([ 1, 2, 3 ])

    numvar = len(bkc)
    numcon = len(bkc)
    numanz = len(asub)

    task.inputdata(numcon, numvar,
                   c, 0.0,
                   aptrb,
                   aptre,
                   asub,
                   aval,
                   bkc,
                   blc,
                   buc,
                   bkc,
                   blx,
                   bux)

    # Set up and input quadratic objective
    qsubi = array([ 0, 1, 2, 2 ])

```

```

qsubj = array([ 0, 1, 0, 2 ])
qval = array([ 2.0, 0.2, -1.0, 2.0 ])

task.putqobj(qsubi,qsubj,qval)

task.putobjsense(mosek.objsense.minimize)

# Optimize
task.optimize()

# Fetch the interior solution
xx = zeros(numvar, Float)
task.getsolutionslice(mosek.soltype.itr,
                      mosek.solitem.xx,
                      0, numvar,
                      xx);

print "Primal interior solution: "
print "\t",xx

return None

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

```
qsubi = array([ 0, 1, 2, 2 ])
qsubj = array([ 0, 1, 0, 2 ])
qval = array([ 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:

```
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 (5.13).

Consider the problem:

$$\begin{aligned} & \text{minimize} && x_1^2 + 0.1x_2^2 + x_3^2 - x_1x_3 - x_2 \\ & \text{subject to} && 1 \leq x_1 + x_2 + x_3 - x_1^2 - x_2^2 - 0.1x_3^2 + 0.2x_1x_3, \\ & && x \geq 0. \end{aligned} \quad (5.18)$$

This is equivalent to

$$\begin{aligned} & \text{minimize} && 1/2x^T Q^o x + c^T x \\ & \text{subject to} && 1/2x^T Q^0 x + Ax \geq b, \end{aligned} \quad (5.19)$$

where

$$Q^o = \begin{bmatrix} 2 & 0 & -1 \\ 0 & 0.2 & 0 \\ -1 & 0 & 2 \end{bmatrix}, \quad c = \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix}, \quad A = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}, \quad b = 1. \quad (5.20)$$

$$Q^0 = \begin{bmatrix} -2 & 0 & 0.2 \\ 0 & -2 & 0 \\ 0.2 & 0 & -0.2 \end{bmatrix}. \quad (5.21)$$

5.3.2.1 Source code

```
##
#   Copyright: Copyright (c) 1998-2007 MOSEK ApS, Denmark. All rights reserved.
#
#   File:      qcqo1.py
#
```

```

# Purpose: Demonstrate how to solve a quadratic
#          optimization problem using the MOSEK API.
#
#          minimize x0^2 + 0.1 x1^2 + x2^2 - x0 x2 - x1
#          s.t 1 <= x0 + x1 + x2 - x0^2 - x1^2 - 0.1 x2^2 + 0.2 x0 x2
#          x >= 0
##

import sys
import os

import pymosek as mosek

from mosekarr import array, Float, zeros, ones

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

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

# We might write everything directly as a script, but it looks nicer
# to create a function.
def main():
    # Open MOSEK and create an environment and task
    # Create a handle to MOSEK
    mskhandle = mosek.mosek()
    # Make a MOSEK environment
    env = mskhandle.Env()
    # Attach a printer to the environment
    env.set_Stream(mosek.streamtype.log, streamprinter)
    # Initialize the environment
    env.init()
    # Create a task
    task = env.Task()

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

    bkc = array([ mosek.boundkey.lo,
                  mosek.boundkey.lo,
                  mosek.boundkey.lo ])
    blx = array([ 0.0, 0.0, 0.0 ])
    bux = array([ inf, inf, inf ])

    c = array([ 0.0, -1.0, 0.0 ])

    asub = array([ 0, 0, 0 ])
    aval = array([ 1.0, 1.0, 1.0 ])
    aptrb = array([ 0, 1, 2 ])
    aptre = array([ 1, 2, 3 ])

```

```

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

task.inputdata(numcon,numvar,
               c,0.0,
               aptrb,
               aptre,
               asub,
               aval,
               bkc,
               blc,
               buc,
               bkx,
               blx,
               bux)

# Set up and input quadratic objective

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

task.putqobj(qsubi,qsubj,qval)

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

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

# put Q^0 in constraint with index 0.

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

task.putobjsense(mosek.objsense.minimize)

# Optimize
task.optimize()

# Fetch the interior solution
xx = zeros(numvar,Float)
task.getsolutionslice(mosek.soltype.itr,
                      mosek.solitem.xx,
                      0,numvar,
                      xx)

print "Primal interior solution: "
print "\t",xx

return None

# call the main function
try:

```

```

    main ()
except mosek.Exception, err:
    print "ERROR: %s" % str(err)
    sys.exit(1)
except:
    import traceback
    traceback.print_exc()
    sys.exit(1)

```

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

$$\begin{aligned}
 & \text{minimize} && c^T x + c^f \\
 & \text{subject to} && l^c \leq Ax \leq u^c, \\
 & && l^x \leq x \leq u^x, \\
 & && x \in \mathcal{C}
 \end{aligned} \tag{5.22}$$

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

MOSEK supports two specific cones apart from the real numbers:

- The quadratic cone:

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

- The rotated quadratic cone:

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

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 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

$$\begin{aligned}
 & \text{minimize} && x_4 + x_5 \\
 & \text{subject to} && x_0 + x_1 + x_2 + x_3 = 1, \\
 & && x_0, x_1, x_2, x_3 \geq 0, \\
 & && x_4 \geq \sqrt{x_0^2 + x_2^2}, \\
 & && x_5 \geq \sqrt{x_1^2 + x_3^2}
 \end{aligned} \tag{5.23}$$

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

```

#
# Copyright: Copyright (c) 1998-2007 MOSEK ApS, Denmark. All rights reserved.
#
# File:      cqo1.py
#
# Purpose:   Demonstrates how to solve a small conic quadratic
#            optimization problem using the MOSEK API.
#

import sys
import os

import pymosek as mosek

try:
    from Numeric import array, Float, zeros, ones
except ImportError:
    from mosekarr import array, Float, zeros, ones

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

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

# We might write everything directly as a script, but it looks nicer
# to create a function.
def main():
    # Create a handle to MOSEK

```

```

mskhandle = mosek.mosek ()
# Make a MOSEK environment
env = mskhandle.Env ()
# Attach a printer to the environment
env.set_Stream (mosek.streamtype.log, streamprinter)
# Initialize the environment
env.init ()

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

bkc = array([ mosek.boundkey.fx ])
blc = array([ 1.0 ])
buc = array([ 1.0 ])

c = array([
            0.0,          0.0,          0.0,
            0.0,          1.0,          1.0 ])
bkc = array([ mosek.boundkey.lo,mosek.boundkey.lo,mosek.boundkey.lo,
            mosek.boundkey.lo,mosek.boundkey.fr,mosek.boundkey.fr ])
blx = array([
            0.0,          0.0,          0.0,
            0.0,          -inf,         -inf ])
bux = array([
            inf,          inf,          inf,
            inf,          inf,          inf ])

asub = array([ 0, 0, 0, 0 ])
acof = array([ 1.0, 1.0, 1.0, 1.0 ])

#      column 0 1 2 3 4 5
ptrb = array([ 0, 1, 2, 3, 4, 4 ])
ptre = array([ 1, 2, 3, 4, 4, 4 ])

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

# Load data into task
task.inputdata (numcon,numvar,
               c,      # linear objective coefficients
               0.0,    # objective fixed value
               ptrb,
               ptre,
               asub,acof,
               bkc,blc,buc,
               bkx,blx,bux)

# Input the cones
task.appendcone(mosek.conetype.quad,
               0.0,
               array([ 4, 0, 2 ]))
task.appendcone(mosek.conetype.quad,
               0.0,
               array([ 5, 1, 3 ]))

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

```

```

# Optimize
task.optimize ()

# Output the solution
task.solutionsummary (mosek.streamtype.msg);

xx = zeros(numvar, Float)
task.getsolutionslice(mosek.soltype.itr,
                      mosek.solitem.xx,
                      0, numvar,
                      xx)

print "x =", xx

return None

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

```

5.4.1.2 Source code comments

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

```

task.appendcone(mosek.conetype.quad,
                0.0,
                array([ 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. c c

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

$$\begin{aligned}
 &\text{maximize} && x_0 + 0.64x_1 \\
 &\text{subject to} && 50x_0 + 31x_1 \leq 250, \\
 & && 3x_0 - 2x_1 \geq -4, \\
 & && x_0, x_1 \geq 0 \quad \text{and integer}
 \end{aligned} \tag{5.24}$$

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

5.5.1.1 Source code

The example (5.24) 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:

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

The complete source for the example is listed below.

```
##
#   Copyright: Copyright (c) 1998-2007 MOSEK ApS, Denmark. All rights reserved.
#
#   File:      mio1.cs
#
#   Purpose:   Demonstrates how to solve a small mixed
#               integer linear optimization problem using the MOSEK Python API.
##

import sys

import pymosek as mosek

from mosekarr import array, Float, zeros, ones

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

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

# We might write everything directly as a script, but it looks nicer
# to create a function.
def main():
    # Create a handle to MOSEK
```

```

mskhandle = mosek.mosek ()
# Make a MOSEK environment
env = mskhandle.Env ()
# Attach a printer to the environment
env.set_Stream (mosek.streamtype.log, streamprinter)
# Initialize the environment
env.init ()

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

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

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

c  = array([          1.0,          0.64 ])

asub = array([ 0, 1, 0, 1 ])
acof = array([ 50.0, 3.0, 31.0, -2.0 ])

ptrb = array([ 0, 2 ])
ptre = array([ 2, 4 ])

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

# Input linear data
task.inputdata(numcon,numvar,
               c,0.0,
               ptrb, ptre, asub, acof,
               bkc, blc, buc,
               bkx, blx, bux)

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

# Define variables to be integers
task.putvartypelist(array([ 0, 1 ]),
                    array([ mosek.variabletype.type_int,
                           mosek.variabletype.type_int ]))
task.putintparam (mosek.iparam.mio_mode,
                  mosek.miomode.satisfied)

# Optimize
task.optimize()

if task.solutiondef(mosek.soltype.itg):

    # Output a solution
    xx = zeros(numvar, Float)
    task.getsolutionslice(mosek.soltype.itg,
                          mosek.solitem.xx,

```

```

                                0, numvar,
                                xx)
    print "x =", xx
else:
    print "Integer solution not defined. Probably a problem with 'mosekglob' optimizer."

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

```

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

$$\begin{aligned}
 &\text{maximize} && 7x_0 + 10x_1 + x_2 + 5x_3 \\
 &\text{subject to} && x_0 + x_1 + x_2 + x_3 \leq 2.5 \\
 &&& x_0, x_1, x_2 \text{ integer, } \quad x_0, x_1, x_2, x_3 \geq 0
 \end{aligned} \tag{5.25}$$

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

```

##
#   Copyright: Copyright (c) 1998-2007 MOSEK ApS, Denmark. All rights reserved.

```

```

#
#   File:      mioinitsol.cs
#
#   Purpose:   Demonstrates how to solve a small mixed
#               integer linear optimization problem using the MOSEK Python API.
##

import sys

import pymosek as mosek

try:
    from Numeric import array, Float, zeros, ones
except ImportError:
    from mosekarr import array, Float, zeros, ones

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

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

# We might write everything directly as a script, but it looks nicer
# to create a function.
def main ():
    # Create a handle to MOSEK
    mskhandle = mosek.mosek ()
    # Make a MOSEK environment
    env = mskhandle.Env ()
    # Attach a printer to the environment
    env.set_Stream (mosek.streamtype.log, streamprinter)
    # Initialize the environment
    env.init ()

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

    bkc = array([ mosek.boundkey.up ])
    blc = array([ -inf,
                  ])
    buc = array([ 2.5
                  ])

    bkc = array([ mosek.boundkey.lo,
                  mosek.boundkey.lo,
                  mosek.boundkey.lo,
                  mosek.boundkey.lo ])

    blx = array([0.0, 0.0, 0.0, 0.0 ])
    bux = array([ inf, inf,  inf,  inf ])

    c   = array([ 7.0, 10.0, 1.0, 5.0 ])

```

```

asub = array([ 0, 0, 0, 0 ])
acof = array([ 1.0, 1.0, 1.0, 1.0])

ptrb = array([ 0, 1, 2, 3 ])
ptre = array([ 1, 2, 3, 4 ])

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

# Input linear data
task.inputdata(numcon,numvar,
               c,0.0,
               ptrb, ptre, asub, acof,
               bkc, blc, buc,
               bkx, blx, bux)

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

# Define variables to be integers
task.putvartypelist(array([ 0, 1, 2 ]),
                    array([ mosek.variabletype.type_int,
                           mosek.variabletype.type_int,
                           mosek.variabletype.type_int]))

# Construct an initial feasible solution from the
# values of the integer valuse specified
task.putintparam(mosek.iparam.mio_construct_sol,
                 mosek.onoffkey.on);

# Set status of all variables to unknown
task.makesolutionstatusunknown(mosek.soltype.itg);

# Assign values 1,1,0 to integer variables
task.putsolutioni (
    mosek.accmode.var,
    0,
    mosek.soltype.itg,
    mosek.stakey.supbas,
    0.0,
    0.0,
    0.0,
    0.0);

task.putsolutioni (
    mosek.accmode.var,
    1,
    mosek.soltype.itg,
    mosek.stakey.supbas,
    2.0,
    0.0,
    0.0,
    0.0);

task.putsolutioni (
    mosek.accmode.var,

```



```

        2,
        mosek.soltype.itg,
        mosek.stakey.supbas,
        0.0,
        0.0,
        0.0,
        0.0);

# Optimize
task.optimize()

if task.solutiondef(mosek.soltype.itg):

    # Output a solution
    xx = zeros(numvar, Float)
    task.getsolutionslice(mosek.soltype.itg,
                          mosek.solitem.xx,
                          0, numvar,
                          xx)

    print "x =", xx
else:
    print "Integer solution not defined. Probably a problem with 'mosekg1b' optimizer."

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

```

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:

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

and

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

The following code loads this problem into the optimization task.

```

# Create a handle to MOSEK
mskhandle = mosek.mosek ()
# Create a MOSEK environment
env = mskhandle.Env ()
# Attach a printer to the environment
env.set_Stream (mosek.streamtype.log, streamprinter)
# Initialize the environment
env.init ()

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

# Bound keys for constraints
bkc = array ([mosek.boundkey.up,
              mosek.boundkey.up,
              mosek.boundkey.up])
# Bound values for constraints
blc = array ([-inf, -inf, -inf])
buc = array ([100000.0, 50000.0, 60000.0])

# Bound keys for variables
bkx = array ([mosek.boundkey.lo,
              mosek.boundkey.lo,
              mosek.boundkey.lo])
# Bound values for variables
blx = array ([ 0.0,  0.0,  0.0])
bux = array ([+inf, +inf, +inf])

# Objective coefficients

```

```

csub = array([ 0, 1, 2 ])
cval = array([ 1.5, 2.5, 3.0 ])

# We input the A matrix column-wise
# asub contains row indexes
asub = array([ 0, 1, 2,
              0, 1, 2,
              0, 1, 2])
# acof contains coefficients
acof = array([ 2.0, 3.0, 2.0,
              4.0, 2.0, 3.0,
              3.0, 3.0, 2.0 ])
# aptrb and aptre contains the offsets into asub and acof where
# columns start and end respectively
aptrb = array([ 0, 3, 6 ])
aptre = array([ 3, 6, 9 ])

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

# Append the constraints
task.append(mosek.accmode.con, numcon)

# Append the variables.
task.append(mosek.accmode.var, numvar)

# Input objective
task.putcfix(0.0)
task.putclist(csub, cval)

# Put constraint bounds
task.putboundslice(mosek.accmode.con,
                  0, numcon,
                  bkc, blc, buc)

# Put variable bounds
task.putboundslice(mosek.accmode.var,
                  0, numvar,
                  bkx, blx, bux)

# Input A non-zeros by columns
for j in range(numvar):
    ptrb, ptre = aptrb[j], aptre[j]
    task.putavec(mosek.accmode.var, j,
                asub[ptrb:ptre],
                acof[ptrb:ptre])

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

# Optimize the task
task.optimize()

# Output a solution
xx = zeros(numvar, Float)

```

```
task.getsolutionslice(mosek.soltype.bas,
                     mosek.solitem.xx,
                     0,numvar,
                     xx)
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.

```
task.putaij(0, 0, 3.0)
```

The problem now has the form:

$$\begin{array}{llllll}
 \text{maximize} & 1.5x_0 & + & 2.5x_1 & + & 3.0x_2 \\
 \text{subject to} & 3x_0 & + & 4x_1 & + & 3x_2 & \leq & 100000, \\
 & 3x_0 & + & 2x_1 & + & 3x_2 & \leq & 50000, \\
 & 2x_0 & + & 3x_1 & + & 2x_2 & \leq & 60000,
 \end{array} \tag{5.28}$$

and

$$x_0, x_1, x_2 \geq 0. \tag{5.29}$$

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.

```
# Append a new variable x_3 to the problem */
task.append(mosek.acemode.var,1)

# Set bounds on new variable
task.putbound(mosek.acemode.var,
              task.getnumvar()-1,
              mosek.boundkey.lo,
              0,
              +inf)

# Change objective
task.putcj(task.getnumvar()-1,1.0)

# Put new values in the A matrix
```

```
acolsub = array([0, 2])
acolval = array([4.0, 1.0])

task.putavec(mosek.accmode.var,
task.getnumvar()-1, # column index
            acolsub,
            acolval)
```

After this operation the problem looks this way:

$$\begin{array}{llllllll}
\text{maximize} & 1.5x_0 & + & 2.5x_1 & + & 3.0x_2 & + & 1.0x_3 \\
\text{subject to} & 3x_0 & + & 4x_1 & + & 3x_2 & + & 4x_3 & \leq & 100000, \\
& 3x_0 & + & 2x_1 & + & 3x_2 & & & \leq & 50000, \\
& 2x_0 & + & 3x_1 & + & 2x_2 & + & 1x_3 & \leq & 60000,
\end{array} \tag{5.30}$$

and

$$x_0, x_1, x_2, x_3 \geq 0. \tag{5.31}$$

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 (5.28) was found and then added a column was added to get (5.30). 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.

```
# Change optimizer to simplex free and reoptimize
task.putintparam(mosek.iparam.optimizer, mosek.optimizertype.free_simplex)
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

$$x_0 + 2x_1 + x_2 + x_3 \leq 30000 \tag{5.32}$$

to the problem which is done in the following code:

```
# Append a new constraint
task.append(mosek.acemode.con,1)

# Set bounds on new constraint
task.putbound(
    mosek.acemode.con,
    task.getnumcon()-1, # row index
    mosek.boundkey.up,
    -inf,
    30000)

# Put new values in the A matrix

arowsub = array([0, 1, 2, 3 ])
arowval = array([1.0, 2.0, 1.0, 1.0])

task.putavec(mosek.acemode.con,
    task.getnumcon()-1, # row index
    arowsub,
    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.putmaxnumanz.** Estimate for the number of non-zeros in A .
- **Task.putmaxnumqnz.** Estimate for the number of non-zeros in the quadratic terms.

None of these functions 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.

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

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

- The quadratic terms in the objective:

$$q_{qosubi[t], qosubj[t]}^o = qoval[t], \quad t = 0, \dots, \text{numqonz} - 1. \quad (5.33)$$

- The linear terms in the objective:

$$c_j = c[j], \quad j = 0, \dots, \text{numvar} - 1 \quad (5.34)$$

- The fixed term in the objective:

$$c^f = \text{cfix}. \quad (5.35)$$

- The quadratic terms in the constraints:

$$q_{qcsubi[t], qcsubj[t]}^{qcsubk[t]} = qcval[t], \quad t = 0, \dots, \text{numqcnz} - 1. \quad (5.36)$$

- The linear terms in the constraints:

$$a_{asub[t], j} = avel[t], \quad t = \text{ptrb}[j], \dots, \text{ptre}[j] - 1, \\ j = 0, \dots, \text{numvar} - 1. \quad (5.37)$$

- 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 `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

$$l_k^c = \text{blc}[k], \quad k = 0, \dots, \text{numcon} - 1 \quad (5.38)$$

and

$$u_k^c = \text{buc}[k], \quad k = 0, \dots, \text{numcon} - 1. \quad (5.39)$$

- 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

$$l_j^x = \text{blx}[j], \quad j = 0, \dots, \text{numvar} - 1. \quad (5.40)$$

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

$$u_j^x = \text{bux}[j], \quad j = 0, \dots, \text{numvar} - 1. \quad (5.41)$$

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

```
c = zeros(numvar,Float)
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

```
upper_bound = zeros(8,Float)
lower_bound = zeros(8,Float)
bound_key   = array([None] * 8)

task.getboundslice(accmode.con, 2,10,
                   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

```
bound_index = array([1, 6, 3, 9])
bound_key   = array([boundkey.fr,boundkey.lo,boundkey.up,boundkey.fx])
upper_bound = array([0.0, -10.0, 0.0, 5.0])
lower_bound = array([0.0, 0.0, 6.0, 5.0])
task.putboundlist(accmode.con, bound_index,
                  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:

```
subi = array([ 1, 3, 5 ])
subj = array([ 2, 3, 4 ])
cof = array([ 1.1, 4.3, 0.2 ])
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 $\text{ptrb}[j]$ is the position of the first value/index in **aval** / **asub** for column j .

ptre: Where $\text{ptre}[j]$ 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

$$a_{\text{asub}[k],j} = \text{aval}[k], \quad k = \text{ptrb}[j], \dots, \text{ptre}[j] - 1. \quad (5.42)$$

As an example consider the matrix

$$A = \begin{bmatrix} 1.1 & & 1.3 & 1.4 & \\ & 2.2 & & & 2.5 \\ 3.1 & & & 3.4 & \\ & & 4.4 & & \end{bmatrix}. \quad (5.43)$$

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

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

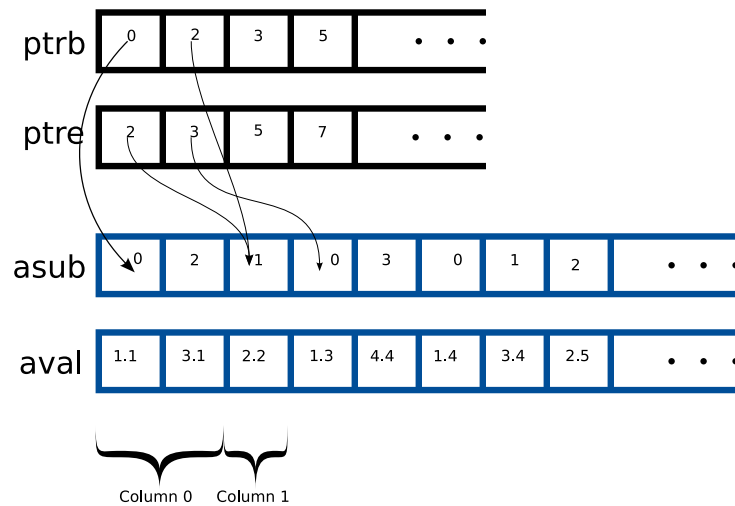


Figure 5.1: The matrix A (5.43) represented in column ordered sparse matrix format.

Fig. 5.1 illustrates how the matrix A (5.43) is represented in column ordered sparse matrix format.

5.8.3.3 Row ordered sparse matrix

The matrix A (5.43) 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:

```
import mosekarr

# Create an array of integers
a0 = mosekarr.array([1,2,3],mosekarr.Int)
# Create an array of floats
a1 = mosekarr.array([1,2,3],mosekarr.Float)
# Create an integer array of ones
a2 = mosekarr.ones(10)
```

```
# Create an float array of ones
a3 = mosekarr.ones(10,mosekarr.Float)
# Create a range of integers 5,6,...,9
a4 = mosekarr.range(5,10)
# Create and array of objects
a5 = mosekarr.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:

```
a = ones(10,mosekarr.Float)
b = 1.0 * range(10)

# element-wise multiplication, addition and subtraction
c0 = a * b
c1 = a + b
c2 = a - b

# multiplly each element by 2.1
c4 = a * 2.1

# add 2 to each element
c5 = a + 2
```

Finally, if the Numeric package is available, arrays can be converted between Numeric and mosekarr types:

```
A = Numeric.array(mosekarr.ones(10))
B = mosekarr.array(Numeric.ones(10))
```

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 and Python Numeric arrays (*not* from the standard `array` module), and it does not implicitly convert arrays. This means that if MOSEK in some function call expects an array of `floats`, it will not accept, say, an array of `ints`. The MOSEK Python default array object is found in the module `mosekarr` which contains a minimal subset of functionality similar to Numeric.

Typically type errors occur in two situations:

- The array-type does not match the expected, and an `ValueError` exception is raised with a message

In function XX: Argument <YY> should be of type array

Note in particular that the python module MA (Matlab compatibility functions) in some cases produce Python arrays, *not* Python Numeric arrays. The type of a variable `xx` can always be displayed by the line

```
print type(xx)
```

- The array had the correct type, but the elements in the array did not have the expected type. This will result in a `ValueError` exception with a message

```
In function XX: Array argument <YY> has wrong type
```

This may be due to implicit typing; for example an array

```
Numeric.array([1,2,3])
```

will become an array of integers while

```
Numeric.array([1.0,2.0,3.0])
```

is an array of floats. To avoid this kind of errors, it is recommended that the array type is explicitly provided, for example as

```
Numeric.array([1.0,2.0,3.0], Numeric.Float)
```

The type-code of an array `xx` can be displayed as

```
print xx.typecode
```

The specific meanings of type-codes are listed in the Python `Numeric` documentation.

Chapter 6

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.

6.1 Linear optimization

A linear optimization problem can be written as

$$\begin{array}{ll} \text{minimize} & c^T x + c^f \\ \text{subject to} & \begin{array}{ll} l^c \leq Ax \leq u^c, \\ l^x \leq x \leq u^x, \end{array} \end{array} \tag{6.1}$$

where

- m is the number of constraints.
- n is the number of decision variables.
- $x \in R^n$ is a vector of decision variables.
- $c \in R^n$ is the linear part of the objective function.
- $A \in R^{m \times n}$ is the constraint matrix.

- $l^c \in R^m$ is the lower limit¹ on the activity for the constraints.
- $u^c \in R^m$ is the upper limit on the activity for the constraints.
- $l^x \in R^n$ is the lower limit on the activity for the variables.
- $u^x \in R^n$ is the upper limit on the activity for the variables.

A primal solution (x) is *(primal) feasible* if it satisfies all constraints in (6.1). If (6.1) has at least one primal feasible solution, then (6.1) is said to be (primal) feasible.

In case (6.1) does not have a feasible solution, the problem is said to be *(primal) infeasible*.

6.1.1 Duality for linear optimization

Corresponding to the primal problem (6.1), there is a dual problem

$$\begin{aligned}
 & \text{maximize} && (l^c)^T s_l^c - (u^c)^T s_u^c \\
 & && + (l^x)^T s_l^x - (u^x)^T s_u^x + c^f \\
 & \text{subject to} && A^T y + s_l^x - s_u^x = c, \\
 & && -y + s_l^c - s_u^c = 0, \\
 & && s_l^c, s_u^c, s_l^x, s_u^x \geq 0.
 \end{aligned} \tag{6.2}$$

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. For example

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

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 (6.2). If (6.2) has at least one feasible solution, then (6.2) 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 (6.1), and

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

is a solution to the corresponding dual problem (6.2). A solution which is both primal and dual feasible is denoted a *primal-dual feasible* *primal-dual* solution.

¹We will use the words “bound” and “limit” interchangeably.

6.1.1.1 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 (6.2) 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.

6.1.1.2 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 such 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 (6.1) 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.

6.1.1.3 Primal infeasible problems

If the problem (6.1) is infeasible (has no feasible solution), MOSEK will report a primal certificate of the infeasibility: The dual solution reported is a certificate of infeasibility, and the primal solution is undefined.

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

$$\begin{aligned} & \text{maximize} && (l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x \\ & \text{subject to} && A^T y + s_l^x - s_u^x = 0, \\ & && -y + s_l^c - s_u^c = 0, \\ & && s_l^c, s_u^c, s_l^x, s_u^x \geq 0. \end{aligned} \tag{6.3}$$

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 (6.3) 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 (6.3) is unbounded, and that its dual is infeasible.

We note that the dual of (6.3) is a problem whose constraints are identical to the constraints of the original primal problem (6.1): If the dual of (6.3) is infeasible, so is the original primal problem.

6.1.1.4 Dual infeasible problems

If the problem (6.2) is infeasible (has no feasible solution), MOSEK will report a dual certificate of the 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

$$\begin{aligned} & \text{minimize} && c^T x \\ & \text{subject to} && Ax - x^c = 0, \\ & && \bar{l}^c \leq x^c \leq \bar{u}^c, \\ & && \bar{l}^x \leq x \leq \bar{u}^x \end{aligned} \tag{6.4}$$

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 (6.4) is unbounded, and that dual of (6.4) is infeasible.

We note that the dual of (6.4) is a problem whose constraints are identical to the constraints of the original dual problem (6.2): If the dual of (6.4) is infeasible, so is the original dual problem.

6.1.2 Primal and dual infeasible case

In case that both the primal problem (6.1) and the dual problem (6.2) are infeasible, MOSEK will report only one of the two possible certificates — which one is not defined (MOSEK returns the first certificate found).

6.2 Linear network flow problems

Network flow problems are a special class of linear optimization problems which has many applications. The class of network flow problems can be specified as follows. Let $G = (\mathcal{N}, \mathcal{A})$ be a directed network. 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. Now the minimum cost of a network flow problem can be stated as follows

$$\begin{aligned} & \text{minimize} && \sum_{(i,j) \in \mathcal{A}} c_{ij} x_{ij} \\ & \text{subject to} && l_i^c \leq \sum_{\{j: (i,j) \in \mathcal{A}\}} x_{ij} - \sum_{\{j: (j,i) \in \mathcal{A}\}} x_{ji} \leq u_i^c \quad \forall i \in \mathcal{N}, \\ & && l_{ij}^x \leq x_{ij} \leq u_{ij}^x \quad \forall (i, j) \in \mathcal{A}. \end{aligned} \quad (6.5)$$

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.

It is well-known that problems with network flow structure can be solved efficiently with a specialized version of the simplex method. MOSEK includes a highly tuned network simplex implementation, see Section 7.3.1 for further details on how to invoke the network optimizer.

6.3 Quadratic and quadratically constrained optimization

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

$$\begin{aligned} & \text{minimize} && \frac{1}{2} x^T Q^o x + c^T x + c^f \\ & \text{subject to} && l_k^c \leq \frac{1}{2} x^T Q^k x + \sum_{j=0}^{n-1} a_{k,i} x_j \leq u_k^c, \quad k = 0, \dots, m-1, \\ & && l^x \leq x \leq u^x, \quad j = 0, \dots, n-1, \end{aligned} \quad (6.6)$$

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 $l_k^c > -\infty$ and $u_k^c < \infty$, 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.

6.3.1 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 6.4.4.1 for further details.

6.3.2 Reformulating as a separable quadratic problem

The simplest quadratic optimization problem is

$$\begin{aligned} & \text{minimize} && 1/2x^T Qx + c^T x \\ & \text{subject to} && Ax = b, \\ & && x \geq 0. \end{aligned} \tag{6.7}$$

The problem (6.7) is said to be a separable problem if Q is a diagonal matrix or, in other words, that 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

$$Q = F^T F. \tag{6.8}$$

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

Using (6.8), the problem (6.7) can be reformulated as

$$\begin{aligned} & \text{minimize} && 1/2y^T Iy + c^T x \\ & \text{subject to} && Ax = b, \\ & && Fx - y = 0, \\ & && x \geq 0. \end{aligned} \tag{6.9}$$

The problem (6.9) is also a quadratic optimization problem and has more constraints and variables than (6.7). 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 dense matrix.

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

$$1/2x^T (F^T F)x \leq b \tag{6.10}$$

where F is a matrix and b is a scalar. (6.10) can be reformulated as

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

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

Next, consider the constraint

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

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, i.e. D is for instance a diagonal or a block diagonal matrix. If this is the case, then it may be worthwhile performing the reformulation

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

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.

6.4 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

$$\begin{aligned} &\text{minimize} && c^T x + c^f \\ &\text{subject to} && l^c \leq Ax \leq u^c, \\ &&& l^x \leq x \leq u^x, \\ &&& x \in \mathcal{C}. \end{aligned} \tag{6.11}$$

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 $x_t \in R^{n_t}$. Please note that the n -dimensional Euclidean space 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

- R set:

$$\mathcal{C}_t = \{x \in R^{n^t}\}.$$

- Quadratic cone:

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

- Rotated quadratic cone:

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

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 6.4.4.

6.4.1 Duality for conic optimization

The dual problem corresponding to the conic optimization problem (6.11) is given by

$$\begin{aligned} & \text{maximize} && (l^c)^T s_l^c - (u^c)^T s_u^c \\ & && + (l^x)^T s_l^x - (u^x)^T s_u^x + c^f \\ & \text{subject to} && A^T y + s_l^x - s_u^x + s_n^x = c, \\ & && -y + s_l^c - s_u^c = 0, \\ & && s_l^c, s_u^c, s_l^x, s_u^x \geq 0, \\ & && s_n^x \in \mathcal{C}^* \end{aligned} \tag{6.12}$$

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

- R set:

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

- Quadratic cone:

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

- Rotated quadratic cone:

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

6.4.2 The dual of the dual

The dual problem corresponding to the dual problem is the primal problem.

6.4.3 Infeasibility

In case MOSEK finds a problem to be infeasible it will report a certificate of the infeasibility. This works exactly as for linear problems (see sections 6.1.1.3 and 6.1.1.4).

6.4.4 Examples

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

6.4.4.1 Quadratic objective and constraints

From Section 6.3.2 we know that any convex quadratic problem can be stated on the form

$$\begin{aligned} & \text{minimize} && 0.5 \|Fx\|^2 + c^T x, \\ & \text{subject to} && 0.5 \|Gx\|^2 + a^T x \leq b, \end{aligned} \tag{6.13}$$

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 (6.13) can be reformulated as

$$\begin{aligned} & \text{minimize} && 0.5 \|t\|^2 + c^T x, \\ & \text{subject to} && 0.5 \|z\|^2 + a^T x \leq b, \\ & && Fx - t = 0, \\ & && Gx - z = 0 \end{aligned} \tag{6.14}$$

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.

$$\begin{aligned} & \text{minimize} && v + c^T x, \\ & \text{subject to} && p + a^T x = b, \\ & && Fx - t = 0, \\ & && Gx - z = 0, \\ & && w = 1, \\ & && q = 1, \\ & && \|t\|^2 \leq 2vw, \quad v, w \geq 0, \\ & && \|z\|^2 \leq 2pq, \quad p, q \geq 0. \end{aligned} \tag{6.15}$$

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

If we assume that F is a non-singular matrix — for instance a diagonal matrix — then

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

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

$$\begin{aligned} & \text{minimize} && v + c^T F^{-1}t, \\ & \text{subject to} && p + a^T F^{-1}t = b, \\ & && VF^{-1}t - z = 0, \\ & && w = 1, \\ & && q = 1, \\ & && \|t\|^2 \leq 2vw, \quad v, w \geq 0, \\ & && \|z\|^2 \leq 2pq, \quad p, q \geq 0. \end{aligned} \tag{6.16}$$

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

6.4.4.2 Minimizing a sum of norms

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

$$\begin{aligned} & \text{minimize} && \sum_{i=1}^k \|x^i\| \\ & \text{subject to} && Ax = b, \end{aligned} \tag{6.17}$$

where

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

This problem is equivalent to

$$\begin{aligned} & \text{minimize} && \sum_{i=1}^k z_i \\ & \text{subject to} && Ax = b, \\ & && \|x^i\| \leq z_i, \quad i = 1, \dots, k, \end{aligned} \tag{6.18}$$

which in turn is equivalent to

$$\begin{aligned} & \text{minimize} && \sum_{i=1}^k z_i \\ & \text{subject to} && Ax = b, \\ & && (z_i, x^i) \in \mathcal{C}_i, \quad i = 1, \dots, k \end{aligned} \tag{6.19}$$

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 (6.19) is

$$\begin{aligned} & \text{maximize} && b^T y \\ & \text{subject to} && A^T y + s = c, \\ & && t_i = 1, \quad i = 1, \dots, k, \\ & && (t_i, s^i) \in \mathcal{C}_i, \quad i = 1, \dots, k \end{aligned} \tag{6.20}$$

where

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

This problem is equivalent to

$$\begin{aligned} & \text{maximize} && b^T y \\ & \text{subject to} && A^T y + s = c, \\ & && \|s^i\|_2^2 \leq 1, \quad i = 1, \dots, k. \end{aligned} \tag{6.21}$$

Please note that the dual problem can be reduced to an “ordinary” convex quadratically constrained optimization problem in this case 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 (6.20) rather than the primal problem (6.19).

6.4.4.3 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.

For example, 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

$$\begin{aligned} & \text{minimize} && c^T x \\ & \text{subject to} && \sum_{j=1}^n z_j = b, \\ & && v_j = \sqrt{2}, \quad j = 1, \dots, n, \\ & && v_j^2 \leq 2z_j x_j, \quad j = 1, \dots, n, \\ & && x, z \geq 0, \end{aligned} \tag{6.22}$$

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 (6.22) 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

$$\begin{aligned} t^2 & \leq 2xz, \\ z & = 0.5, \\ x, z, & \geq 0. \end{aligned} \tag{6.23}$$

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

$$\begin{aligned} x^{1.5} & \leq t, \\ 0 & \leq x. \end{aligned} \tag{6.24}$$

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

$$\begin{aligned} x^2 & \leq 2st, \\ s, t & \geq 0. \end{aligned} \tag{6.25}$$

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

$$\begin{aligned} x^2 &\leq 2st, \\ w^2 &\leq 2vr, \\ x &= v, \\ s &= w, \\ r &= \frac{1}{8}, \\ s, t, v, r &\geq 0, \end{aligned} \tag{6.26}$$

then

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

Moreover,

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

leading to the conclusion

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

(6.26) is a conic reformulation which is equivalent to (6.24). Please note that the $x \geq 0$ constraint does not appear explicitly in (6.25) and (6.26), but implicitly since $x = v \geq 0$.

Finally, it should be mentioned that any polynomial term of form x^g where g is a positive rational number can be represented using conic quadratic constraints [3, pp. 12-13]

6.4.4.4 Further reading

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

6.4.5 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.

6.4.5.1 Non-attainment in the primal problem

Consider the example

$$\begin{array}{ll} \text{minimize} & z \\ \text{subject to} & 2yz \geq x^2, \\ & x = \sqrt{2}, \\ & y, z \geq 0. \end{array} \quad (6.27)$$

which corresponds to the problem

$$\begin{array}{ll} \text{minimize} & \frac{1}{y} \\ \text{subject to} & y \geq 0. \end{array} \quad (6.28)$$

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

6.4.5.2 Non-attainment in the dual problem

Next, consider the example

$$\begin{array}{ll} \text{minimize} & x_4 \\ \text{subject to} & x_3 + x_4 = 1, \\ & x_1 = 0, \\ & x_2 = 1, \\ & 2x_1x_2 \geq x_3^2, \\ & x_1x_2 \geq 0. \end{array} \quad (6.29)$$

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 (6.29) is

$$\begin{array}{ll} \text{maximize} & y_1 + y_3 \\ \text{subject to} & y_2 + s_1 = 0, \\ & y_3 + s_2 = 0, \\ & y_1 + s_3 = 0, \\ & y_1 = 1, \\ & 2s_1s_2 \geq s_3^2, \\ & s_1s_2 \geq 0. \end{array} \quad (6.30)$$

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, there no primal and dual bounded optimal solution that has zero duality gap exists. Of course it is possible to find a primal and dual feasible solution such that the duality gap is close to zero, however, s_1^* will be very large (unless a large duality gap is allowed). This is likely to make the problem (6.29) hard to solve.

An inspection of problem (6.29) 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 (6.29) or eliminate x_1 and x_3 from the problem it becomes easy to solve.

6.5 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 leading to the introduction of 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 are either 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 might be a good idea to introduce more constraints and variables if it makes the model separable. Also a big but sparse problem might 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].

6.5.1 Avoid nearly infeasible models

Consider the linear optimization problem

$$\begin{aligned}
 & \text{minimize} \\
 & \text{subject to} \quad \begin{aligned} x + y &\leq 10^{-10} + \alpha, \\ 1.0e4x + 2.0e4y &\geq 10^{-6}, \\ x, y &\geq 0. \end{aligned}
 \end{aligned} \tag{6.31}$$

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.

6.6 Examples continued

6.6.1 The absolute value

Assume we have a constraint for the form

$$|f^T x + g| \leq b \tag{6.32}$$

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

It is easy to verify that the constraint (6.32) is equivalent to

$$-b \leq f^T x + g - t \leq b \tag{6.33}$$

which is a set of ordinary linear inequality constraints.

Please note that equalities involving and absolute value such as

$$|x| = 1$$

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

6.6.2 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 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 .

6.6.2.1 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

$$\begin{aligned} & \text{minimize} && V[r^T(w^0 + x)] \\ & \text{subject to} && E[r^T(w^0 + x)] = t, \\ & && e^T x = 0, \end{aligned} \tag{6.34}$$

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

$$\begin{aligned} & \text{minimize} && (w^0 + x)^T \Sigma (w^0 + x) \\ & \text{subject to} && \bar{r}^T (w^0 + x) = t, \\ & && e^T x = 0, \end{aligned} \tag{6.35}$$

6.6.2.2 Conic quadratic reformulation.

An equivalent conic quadratic reformulation is given by:

$$\begin{aligned} & \text{minimize} && f \\ & \text{subject to} && \Sigma^{\frac{1}{2}}(w^0 + x) - g = 0, \\ & && \bar{r}^T (w^0 + x) = t, \\ & && e^T x = 0, \\ & && f \geq \|g\|. \end{aligned} \tag{6.36}$$

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.

6.6.2.3 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 modelled as follows

$$\text{commission} + \frac{\text{bid}}{\text{ask}} - \text{spread} + \theta \sqrt{\frac{\text{trade volume}}{\text{daily volume}}}, \tag{6.37}$$

and that these are important to incorporate 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:

$$\begin{aligned} & \text{minimize} && f \\ & \text{subject to} && \Sigma^{\frac{1}{2}}(w^0 + x) - g = 0, \\ & && \bar{r}^T(w^0 + x) = t, \\ & && e^T x + e^T y = 0, \\ & && |x_j|(m_j|x_j|^{\frac{1}{2}}) \leq y_j, \\ & && f \geq \|g\|. \end{aligned} \tag{6.38}$$

Now, defining the variable transformation

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

we obtain

$$\begin{aligned} & \text{minimize} && f \\ & \text{subject to} && \Sigma^{\frac{1}{2}}(w^0 + x) - g = 0, \\ & && \bar{r}^T(w^0 + x) = t, \\ & && e^T x + m^T \bar{y} = 0, \\ & && |x_j|^{3/2} \leq \bar{y}_j, \\ & && f \geq \|g\|. \end{aligned} \tag{6.39}$$

As shown in Section 6.4.4.3 the set

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

can be modelled by

$$\begin{aligned} x_j & \leq z_j, \\ -x_j & \leq z_j, \\ z_j^2 & \leq 2s_j \bar{y}_j, \\ u_j^2 & \leq 2v_j q_j, \\ z_j & = v_j, \\ s_j & = u_j, \\ q_j & = \frac{1}{8}, \\ q_j, s_j, \bar{y}_j, v_j, q_j & \geq 0. \end{aligned} \tag{6.40}$$

6.6.2.4 Further reading

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

Chapter 7

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.

7.1 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: Solving the actual optimization.

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.

7.1.1 Presolve

Before an optimizer actually performs the optimization the problem is normally 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 the presolve is known to be unable to reduce the size of a problem significantly, then turning off the presolve is beneficial. 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 usually

- the eliminator, and
- the linear dependency check.

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

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. By implied free variable is meant that the constraint $y \geq 0$ is redundant and hence y can be treated as a free variable.

For large scale problems the eliminator usually removes many constraints and variables. However, in some cases few or no eliminations can be performed and moreover, the eliminator may consume a lot of memory and time. If this is the case it is worthwhile to disable the eliminator by setting the parameter `mosek.iparam.presolve_eliminator_use` to `mosek.onoffkey.off`.

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, i.e. one of the constraints is redundant. 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, and therefore it is strongly recommended to build models without linear dependencies. In case the linear dependencies are removed at the modelling stage, the linear dependency check can safely be disabled by setting the parameter `mosek.iparam.presolve_lindep_use` to `mosek.onoffkey.off`.

7.1.2 Dualizer

It is well-known that all linear, conic, and convex optimization problems have an associated dual problem. Moreover, even if the dual instead of the primal problem is solved, it is possible to recover the solution to the original primal problem.

In general, it is very hard to say whether it is easier to solve the primal or the dual problem but MOSEK has some heuristics for deciding which of the two problems to solve. Which form of the problem (primal or dual) that is solved is displayed in the MOSEK log. Please note that the dualizer is transparent, and all solution values returned by the optimizer refer to the original primal problem.

The dualizer can be controlled 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.

Finally, please note that currently only linear problems may be dualized.

7.1.3 Scaling

Problems containing data with large and/or small coefficients, say $1.0\text{e}+9$ or $1.0\text{e}-7$, are often hard to solve. Significant digits might 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.

7.1.4 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.

7.2 Linear optimization

7.2.1 Optimizer selection

For linear optimization problems two different types of optimizers are available. The default for linear problems is an interior-point optimizer, however, as an alternative the simplex optimizer can be employed.

The curious reader can consult [21] for a discussion about interior-point and simplex algorithms.

7.2.2 The interior-point optimizer

The MOSEK interior-point optimizer is an implementation of the homogeneous and self-dual algorithm. For a detailed description of the algorithm, please see [10].

7.2.2.1 Basis identification

It is well-known that 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 is 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.

Parameter name	Purpose
<code>mosek.dparam.intpnt.tol.pfeas</code>	Controls primal feasibility.
<code>mosek.dparam.intpnt.tol.dfeas</code>	Controls dual feasibility.
<code>mosek.dparam.intpnt.tol.rel_gap</code>	Controls relative gap.
<code>mosek.dparam.intpnt.tol.infeas</code>	Controls when the problem is declared primal or dual infeasible.
<code>mosek.dparam.intpnt.tol.mu_red</code>	Controls when the complementarity is reduced enough.

Table 7.1: Parameters employed in termination criterion.

7.2.2.2 Interior-point termination criterion

The parameters in Table 7.1 control when the interior-point optimizer terminates.

7.2.3 The simplex based optimizer

An alternative to the interior-point optimizer is the simplex optimizer. The simplex optimizer employs a different approach than the interior-point optimizer when solving a problem. Contrary to the interior-point optimizer the simplex optimizer can exploit a guess for the optimal solution to reduce solution time. Depending on the problem it may be faster or slower to exploit a guess for the optimal solution. See Section 7.2.4 for a discussion.

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

7.2.3.1 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 (6.1) and (6.2) 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`.

7.2.3.2 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 given the 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.

7.2.3.3 Numerical difficulties in the simplex optimizers

MOSEK is designed to minimize numerical difficulties, however, in rare cases the optimizer may have a hard time solving a problem. MOSEK counts a numerical unexpected behavior inside the optimizer as a “set-back”. The user can define how many set-backs the optimizer accepts, and 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.

What counts as a set-back? It is hard to say without getting very technical but obvious cases are repeated singularities when factorizing the basis matrix, repeated loss of feasibility, degeneracy problems (no progress in objective) or 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. If a lot of set-backs still occur, then 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.

7.2.4 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.

7.2.5 The primal or the dual simplex variant?

MOSEK provides both a primal and a dual simplex optimizer. Predicting which simplex optimizer is faster is simply impossible, however, in recent years the dual optimizer has experienced 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 7.6.3.

7.3 Linear network optimization

7.3.1 Network flow problems

Linear optimization problems with the network flow structure specified in Section 6.2 can in most cases be solved significantly faster with a specialized version of the simplex method [2], rather than with the general solvers.

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

To use the network simplex optimizer, do the following

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

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

7.3.2 Embedded network 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*. If the procedure described above is applied, MOSEK will try to exploit this structure to speed up the optimization.

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

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 7.2: Parameters employed in termination criterion.

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.

7.4 Conic optimization

7.4.1 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].

7.4.1.1 Interior-point termination criteria

The parameters controlling when the conic interior-point optimizer terminates are shown in Table 7.2.

7.5 Nonlinear convex optimization

7.5.1 The interior-point optimizer

For quadratic, quadratically constrained, and general convex optimization problems only 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].

7.5.1.1 Interior-point termination criteria

The parameters controlling when the general convex interior-point optimizer terminates are shown in Table 7.3.

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 7.3: Parameters employed in termination criteria.

7.6 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.

7.6.1 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.

7.6.2 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.

7.6.3 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

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 7.4: Default priorities for optimizer selection in concurrent optimization.

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.

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 7.6.3. 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.

7.6.3.1 Concurrent optimization through the API

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

```
##
#   Copyright: Copyright (c) 1998-2007 MOSEK ApS, Denmark. All rights reserved.
#
#   File:      concurrent1.py
#
#   Purpose:   To demonstrate how to optimize in parallel using the
#               concurrent optimizer.
```

```

##

import sys

import pymosek as mosek

try:
    from Numeric import array, Float
except ImportError:
    from mosekarr import array, Float

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

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

# We might write everything directly as a script, but it looks nicer
# to create a function.
def main (args):
    # Open MOSEK and create an environment and task
    # Create a handle to MOSEK
    mskhandle = mosek.mosek ()
    # Create a MOSEK environment
    env = mskhandle.Env ()
    # Attach a printer to the environment
    env.set_Stream (mosek.streamtype.log, streamprinter)
    # Initialize the environment
    env.init ()

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

    task.readdata(args[0])
    task.putintparam(mosek.iparam.optimizer,
                    mosek.optimizertype.concurrent)

    task.putintparam(mosek.iparam.concurrent_num_optimizers, 2)

    task.optimize()

    task.solutionsummary(mosek.streamtype.msg)

# call the main function
try:
    main (sys.argv[1:])
except mosek.Exception, (code,msg):
    print "ERROR: %s" % str(code)
    if msg is not None:
        print "\t%s" % msg

```

```

        sys.exit(1)
    except:
        import traceback
        traceback.print_exc()
        sys.exit(1)

```

7.6.4 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 application to assign arbitrary values to the parameters of each task, 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.

```

##
# Copyright: Copyright (c) 1998-2007 MOSEK ApS, Denmark. All rights reserved.
#
# File:      concurrent2.py
#
# Purpose:   To demonstrate a more flexible interface for concurrent optimization.
##

import sys

import pymosek as mosek

try:
    from Numeric import array, Float
except ImportError:
    from mosekarr import array, Float

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

# Define a stream printer to grab output from MOSEK
class streamprinter:
    def __init__(self, prefix):
        self.prefix = str(prefix)
    def __call__(self, text):
        #sys.stdout.write (self.prefix + text)
        sys.stdout.write (self.prefix + text)
        sys.stdout.flush()
        pass

# We might write everything directly as a script, but it looks nicer
# to create a function.

```

```

def main (args):
    # Open MOSEK and create an environment and task
    # Create a handle to MOSEK
    mskhandle = mosek.mosek ()
    # Create a MOSEK environment
    env = mskhandle.Env ()
    # Attach a printer to the environment
    env.set_Stream (mosek.streamtype.log, streamprinter("[env]"))
    # Initialize the environment
    env.init ()

    # Create a task
    task = env.Task(0,0)
    # Attach a printer to the task
    task.set_Stream (mosek.streamtype.log, streamprinter("simplex: "))

    # Create a task
    task_list = array([env.Task(0,0)])
    # Attach a printer to the task
    task_list[0].set_Stream(mosek.streamtype.log, streamprinter("intpnt: "))

    task.readdata(args[0]);

    # Assign different parameter values to each task.
    # In this case different optimizers.
    task.putintparam(mosek.iparam.optimizer,
                    mosek.optimizertype.primal_simplex)

    task_list[0].putintparam(mosek.iparam.optimizer,
                             mosek.optimizertype.intpnt)

    # Optimize task and task_list[0] in parallel.
    # The problem data i.e. C, A, etc.
    # is copied from task to task_list[0].
    task.optimizeconcurrent(task_list)

    task.solutionsummary(mosek.streamtype.log)

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

```


Chapter 8

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.

8.1 Some notation

In general, an integer optimization problem has the form

$$\begin{aligned} z^* = \quad & \text{minimize} && c^T x \\ & \text{subject to} && \begin{array}{lll} l^c & \leq & Ax & \leq & u^c, \\ l^x & \leq & Ax & \leq & u^x, \\ & & x_j \in \mathcal{Z}, & & \forall j \in \mathcal{J}, \end{array} \end{aligned} \tag{8.1}$$

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

$$\begin{aligned} \underline{z} = & \text{minimize} && c^T x \\ & \text{subject to} && \begin{array}{l} l^c \leq Ax \leq u^c, \\ l^x \leq Ax \leq u^x \end{array} \end{aligned} \quad (8.2)$$

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 (8.1) 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 $\bar{z} - \underline{z}$.

8.2 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.

8.3 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.

8.3.1 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 .

8.3.2 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.

8.3.3 The optimization phase

This phase solves the problem using the branch and cut algorithm.

8.4 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 (8.2), 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 (8.1).

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 8.1: 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.

Table 8.2: 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 8.1. 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 8.1. In Table 8.2 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.

8.5 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 8.4 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].

Chapter 9

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 10 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.

9.1 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 9.1.

The problem represented in figure 9.1 is infeasible, since the total demand

$$2300 = 1100 + 200 + 500 + 500 \tag{9.1}$$

exceeds the total supply

$$2200 = 200 + 1000 + 1000 \tag{9.2}$$

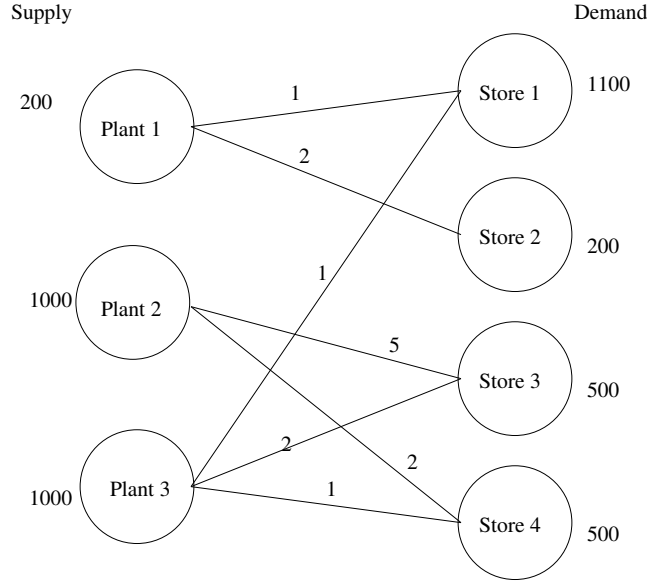


Figure 9.1: Supply, demand and cost of transportation.

If we denote the number of transported goods from plant i to store j by x_{ij} , the problem can be formulated as the LP:

$$\begin{aligned}
 & \text{minimize} && x_{11} &+& 2x_{12} &+& 5x_{23} &+& 2x_{24} &+& x_{31} &+& 2x_{33} &+& x_{34} \\
 & \text{subject to} && x_{11} &+& x_{12} &&&&&&&&&&&&&&& \leq 200, \\
 & &&&&&& x_{23} &+& x_{24} &&&&&&&&&&&& \leq 1000, \\
 & &&&&&&&&&& x_{31} &+& x_{33} &+& x_{34} &&&&&&& \leq 1000, \\
 & && x_{11} &&&&&&&& &+& x_{31} &&&&&&&&& = 1100, \\
 & &&& x_{12} &&&&&&&&&&&&&&&&&& = 200, \\
 & &&&&& x_{23} &+&&&&&&&&& x_{33} &&&&&& = 500, \\
 & &&&&&& x_{24} &+&&&&&&&&&& x_{34} &&& = 500, \\
 & &&&&&&&&&&&&&&&&&&&& x_{ij} \geq 0.
 \end{aligned} \tag{9.3}$$

Solving the problem (9.3) using MOSEK will result in a solution, a solution status and a problem status. Among the log output from the execution of MOSEK on the above problem are the lines:

```

Basic solution
Problem status : PRIMAL_INFEASIBLE
Solution status : PRIMAL_INFEASIBLE_CER

```

The first line indicates that the problem status is primal infeasible. The second line says that a *certificate of the infeasibility* was found. The certificate is returned in place of the solution to the problem.

9.1.1 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 9.1.3) 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

$$x_{12} = 200 \tag{9.4}$$

makes the problem feasible.

9.1.2 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:

$$\begin{array}{ll} \text{minimize} & x_1 \\ \text{subject to} & x_1 \leq 5 \end{array} \tag{9.5}$$

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.

9.1.2.1 A cautious note

The problem

$$\begin{aligned} & \text{minimize} && 0 \\ & \text{subject to} && 0 \leq x_1, \\ & && x_j \leq x_{j+1}, \quad j = 1, \dots, n-1, \\ & && x_n \leq -1 \end{aligned} \tag{9.6}$$

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.

9.1.3 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.

9.1.3.1 Example: Primal infeasibility

We will reuse the example (9.3) 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 executing the command

```
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. Please note that this optimization problem is identical to (9.3), 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**. However, since the reduced problem is usually smaller, it should be easier to locate the cause of the infeasibility in this rather than in the original problem (9.3).

9.1.3.2 Example: Dual infeasibility

The example problem

```

minimize - 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

y1=-1, y2=-1, y3=0, y4=1, y5=1

is a certificate of dual infeasibility. In this example the following infeasibility report is produced (slightly edited):

he 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

$$c^t x^* > 0. \quad (9.7)$$

For a minimization problem this inequality would have been reversed — see (9.19).

From the infeasibility report we see that the variable y4, and the constraints x11 and x33 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 might do one the the following things:

- Put a lower bound in y3. This will directly invalidate the certificate of dual infeasibility.
- Increase the object coefficient of y3. Changing the coefficients sufficiently will invalidate the inequality (9.7) and thus the certificate.
- Put lower bounds on x11 or x31. 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.

9.2 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

$$\begin{aligned} & \text{minimize} && c^T x + c^f \\ & \text{subject to} && \begin{array}{ccc} l^c & \leq & Ax & \leq & u^c, \\ l^x & \leq & x & \leq & u^x \end{array} \end{aligned} \quad (9.8)$$

where the corresponding dual problem is

$$\begin{aligned} & \text{maximize} && (l^c)^T s_l^c - (u^c)^T s_u^c \\ & && + (l^x)^T s_l^x - (u^x)^T s_u^x + c^f \\ & \text{subject to} && \begin{array}{rcl} A^T y + s_l^x - s_u^x & = & c, \\ -y + s_l^c - s_u^c & = & 0, \\ s_l^c, s_u^c, s_l^x, s_u^x & \geq & 0. \end{array} \end{aligned} \quad (9.9)$$

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

$$l_j^x = -\infty \Rightarrow (s_l^x)_j = 0 \quad (9.10)$$

9.2.1 Certificat of primal infeasibility

A certificate of primal infeasibility is *any* solution to the homogenized dual problem

$$\begin{aligned} & \text{maximize} && (l^c)^T s_l^c - (u^c)^T s_u^c \\ & && + (l^x)^T s_l^x - (u^x)^T s_u^x \\ & \text{subject to} && \begin{array}{rcl} A^T y + s_l^x - s_u^x & = & 0, \\ -y + s_l^c - s_u^c & = & 0, \\ s_l^c, s_u^c, s_l^x, s_u^x & \geq & 0. \end{array} \end{aligned} \quad (9.11)$$

with a positive objective value. That is, $(s_l^{c*}, s_u^{c*}, s_l^{x*}, s_u^{x*})$ is a certificat of primal infeasibility if

$$(l^c)^T s_l^{c*} - (u^c)^T s_u^{c*} + (l^x)^T s_l^{x*} - (u^x)^T s_u^{x*} > 0 \quad (9.12)$$

and

$$\begin{aligned} A^T y + s_l^{x*} - s_u^{x*} &= 0, \\ -y + s_l^{c*} - s_u^{c*} &= 0, \\ s_l^{c*}, s_u^{c*}, s_l^{x*}, s_u^{x*} &\geq 0. \end{aligned} \quad (9.13)$$

The well-known Farkas Lemma tells us that (9.8) is infeasible if and only if a certificat of primal infeasibility exists.

Let $(s_l^{c*}, s_u^{c*}, s_l^{x*}, s_u^{x*})$ be a certificate of primal infeasibility then

$$(s_l^{c*})_i > 0 \quad ((s_u^{c*})_i > 0) \quad (9.14)$$

implies that the lower (upper) bound on the i th constraint is important for the infeasibility. Furthermore,

$$(s_l^{x*})_j > 0 \quad ((s_u^{x*})_j > 0) \quad (9.15)$$

implies that the lower (upper) bound on the j th variable is important for the infeasibility.

9.2.2 Certificat of dual infeasibility

A certificate of dual infeasibility is *any* solution to the problem

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & \begin{array}{l} \bar{l}^c \leq Ax \leq \bar{u}^c, \\ \bar{l}^x \leq x \leq \bar{u}^x \end{array} \end{array} \quad (9.16)$$

with negative objective value, where we use the definitions

$$\bar{l}_i^c := \begin{cases} 0, & l_i^c > -\infty, \\ -\infty, & \text{otherwise,} \end{cases} \quad \bar{u}_i^c := \begin{cases} 0, & u_i^c < \infty, \\ \infty, & \text{otherwise,} \end{cases} \quad (9.17)$$

and

$$\bar{l}_i^x := \begin{cases} 0, & l_i^x > -\infty, \\ -\infty, & \text{otherwise,} \end{cases} \quad \text{and} \quad \bar{u}_i^x := \begin{cases} 0, & u_i^x < \infty, \\ \infty, & \text{otherwise.} \end{cases} \quad (9.18)$$

Stated differently, a certificate of dual infeasibility is any x^* such that

$$\begin{array}{lll} c^T x^* & < & 0, \\ \bar{l}^c & \leq & Ax^* \leq \bar{u}^c, \\ \bar{l}^x & \leq & x^* \leq \bar{u}^x \end{array} \quad (9.19)$$

The well-known Farkas Lemma tells us that (9.9) is infeasible if and only if a certificate of dual infeasibility exists.

Observe that if x^* is a certificate of dual infeasibility then for any j such that

$$x_j^* \neq 0, \quad (9.20)$$

variable j is involved in the dual infeasibility.

Chapter 10

Primal feasibility repair

Section 9.1.1 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.

10.1 The main idea

Consider the linear optimization problem with m constraints and n variables

$$\begin{array}{ll} \text{minimize} & c^T x + c^f \\ \text{subject to} & l^c \leq Ax \leq u^c, \\ & l^x \leq x \leq u^x, \end{array} \quad (10.1)$$

which we assume is infeasible. Moreover, we assume that

$$(l^c)_i \leq (u^c)_i, \quad \forall i \quad (10.2)$$

and

$$(l^x)_j \leq (u^x)_j, \quad \forall j \quad (10.3)$$

because otherwise the problem (10.1) 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:

- $w_l^c \in R^m$ (associated with l^c),
- $w_u^c \in R^m$ (associated with u^c),
- $w_l^x \in R^n$ (associated with l^x),
- $w_u^x \in R^n$ (associated with u^x),

Now, the problem

$$\begin{aligned}
 & \text{minimize} && p \\
 & \text{subject to} && l^c \leq Ax + v_l^c - v_u^c \leq u^c, \\
 & && l^x \leq x + v_l^x - v_u^x \leq u^x, \\
 & && (w_l^c)^T v_l^c + (w_u^c)^T v_u^c + (w_l^x)^T v_l^x + (w_u^x)^T v_u^x - p \leq 0, \\
 & && v_l^c, v_u^c, v_l^x, v_u^x \geq 0
 \end{aligned} \tag{10.4}$$

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

$$(w_l^c)^T v_l^c + (w_u^c)^T v_u^c + (w_l^x)^T v_l^x + (w_u^x)^T v_u^x - p \leq 0, \tag{10.5}$$

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 (10.4) is always feasible given the assumptions (10.2) and (10.3).

Please note that if a weight is negative then the resulting problem (10.4) 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 (10.4) and optimize it. Next inspect the optimal solution $(v_l^c)^*$, $(v_u^c)^*$, $(v_l^x)^*$, and $(v_u^x)^*$ to problem (10.4). This solution provides a suggested relaxation of the bounds that will make the problem feasible.

Assume that p^* is an optimal objective value to (10.4). An extension of the idea presented above is to solve the problem

$$\begin{aligned}
 & \text{minimize} && c^T x \\
 & \text{subject to} && l^c \leq Ax + v_l^c - v_u^c \leq u^c, \\
 & && l^x \leq x + v_l^x - v_u^x \leq u^x, \\
 & && (w_l^c)^T v_l^c + (w_u^c)^T v_u^c + (w_l^x)^T v_l^x + (w_u^x)^T v_u^x - p \leq 0, \\
 & && p \\
 & && v_l^c, v_u^c, v_l^x, v_u^x \geq 0
 \end{aligned} \tag{10.6}$$

which minimizes the true objective while making sure that total weighted violations of the bounds is minimal, i.e. equals to p^* .

10.2 Feasibility repair in MOSEK

MOSEK includes functionality that help you construct the problem (10.4) 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.

10.2.1 Usage of negative weights

As the problem (10.4) 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.

10.2.2 Automatical naming

MOSEK can automatically create a new problem of the form (10.4) 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 (10.5) 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 (10.5) is given the name `WSUMVIOLCON`.

The substring “WSUMVIOL” can be replaced by a different string by changing the `mosek.sparam.feasrepair_name_wsumviol` parameter.

10.2.3 Feasibility repair using the API

The `Task.relaxprimal` function takes an existing problem as input and creates a new task containing the problem (10.4). Moreover, if requested this function can solve the problems (10.4) and (10.6) automatically.

The parameter `mosek.iparam.feasrepair_optimize` controls which problem is solved. Its value is used as follows:

- `mosek.feasrepairtype.optimize_none`: The problem (10.4) is constructed, but not solved.
- `mosek.feasrepairtype.optimize_penalty`: The problem (10.4) is constructed and solved.
- `mosek.feasrepairtype.optimize_combined`: The problem (10.6) is constructed and solved.

For further details, please see the description of the function `Task.relaxprimal` in the reference.

10.2.4 An example

Consider this example of linear optimization

$$\begin{array}{llllll}
 \text{minimize} & -10x_1 & & -9x_2, & & \\
 \text{subject to} & 7/10x_1 & + & 1x_2 & \leq & 630, \\
 & 1/2x_1 & + & 5/6x_2 & \leq & 600, \\
 & 1x_1 & + & 2/3x_2 & \leq & 708, \\
 & 1/10x_1 & + & 1/4x_2 & \leq & 135, \\
 & x_1, & & x_2 & \geq & 0. \\
 & & & & & x_2 \geq 650
 \end{array} \tag{10.7}$$

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:

```
#
# Copyright: Copyright (c) 1998-2007 MOSEK ApS, Denmark. All rights reserved.
#
# File:      feasrepairex1.py
#
# Purpose:   To demonstrate how to use the MSK_relaxprimal function to
#            locate the cause of an infeasibility.
#
# Syntax: On command line
#          feasrepairex1 feasrepair.lp
#          feasrepair.lp is located in mosek\<version>\tools\examples.
```

```

import sys

import pymosek as mosek

try:
    from Numeric import array, Float, zeros, ones
except ImportError:
    from mosekarr import array, Float, zeros, ones

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

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

def formatdarray(msk,a):
    r = []
    for v in a:
        if msk.isinfinity(v):
            r.append('inf')
        elif msk.isinfinity(-v):
            r.append('-inf')
        else:
            r.append(str(v))
    return ', '.join(r)

# We might write everything directly as a script, but it looks nicer
# to create a function.
def main (inputfile):
    # Create a handle to MOSEK
    mskhandle = mosek.mosek ()
    # Make a MOSEK environment
    env = mskhandle.Env ()
    # Attach a printer to the environment
    env.set_Stream (mosek.streamtype.log, streamprinter)
    # Initialize the environment
    env.init ()

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

    # Read data
    task.readdata(inputfile)

    task.putintparam(mosek.iparam.feasrepair_optimize,
                    mosek.feasrepairtype.optimize_penalty)

    # Relax
    wlc = array([ 1.0, 1.0, 1.0, 1.0 ])

```

```

wuc = array([ 1.0, 1.0, 1.0, 1.0 ])
wlc = array([ 1.0, 1.0 ])
wux = array([ 1.0, 1.0 ])

relaxed_task = task.relaxprimal(wlc,
                                wuc,
                                wlx,
                                wux);

sum_violation = relaxed_task.getprimalobj (mosek.soltype.bas)

print 'lbc =', formatdarray(mskhandle,wlc)
print 'ubc =', formatdarray(mskhandle,wuc)
print 'lbx =', formatdarray(mskhandle,wlx)
print 'ubx =', formatdarray(mskhandle,wux)

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

```

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 11

Sensitivity analysis

11.1 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. For instance, 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.

11.2 Restrictions

Currently, sensitivity analysis is only available for continuous linear optimization problems. Moreover, MOSEK can only deal with perturbations in bounds and objective coefficients.

11.3 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.

11.4 Sensitivity analysis for linear problems

11.4.1 The optimal objective value function

Assume that we are given the problem

$$\begin{aligned} z(l^c, u^c, l^x, u^x, c) = & \text{minimize} && c^T x \\ & \text{subject to} && l^c \leq Ax \leq u^c, \\ & && l^x \leq x \leq u^x, \end{aligned} \quad (11.1)$$

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

$$\begin{aligned} f_{l_i^c}(\beta) = & \text{minimize} && c^T x \\ & \text{subject to} && l^c + \beta e_i \leq Ax \leq u^c, \\ & && l^x \leq x \leq u^x, \end{aligned} \quad (11.2)$$

where e_i is the i th column of the identity matrix. The function

$$f_{l_i^c}(\beta) \quad (11.3)$$

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 (11.3) shows the optimal objective value as a function of l_i^c .

It is possible to prove that the function (11.3) is a piecewise linear and convex function, i.e. the function may look like the illustration in Figure 11.1.

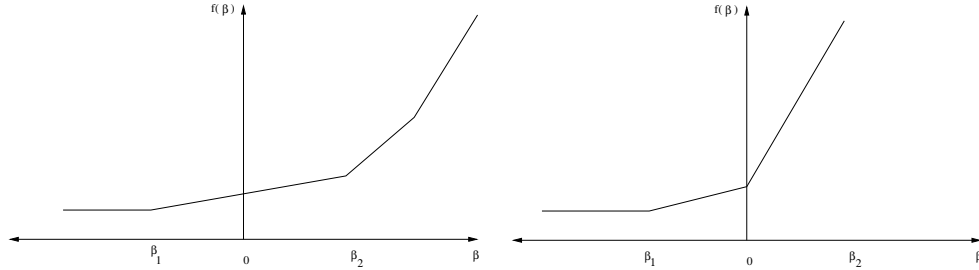


Figure 11.1: The optimal value function $f_{l_i^c}(\beta)$. Left: $\beta = 0$ is in the interior of linearity interval. Right: $\beta = 0$ is a breakpoint.

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

$$f'_{l_i^c}(0), \quad (11.4)$$

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*

$$\beta \in [\beta_1, \beta_2] \quad (11.5)$$

for which

$$f'_{l_i^c}(\beta) = f'_{l_i^c}(0). \quad (11.6)$$

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 11.1. 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 (11.1) as well:

$$\begin{aligned} f_{u_i^c}(\beta) &= z(l^c, u^c + \beta e_i, l^x, u^x, c), & i = 1, \dots, m, \\ f_{l_j^x}(\beta) &= z(l^c, u^c, l^x + \beta e_j, u^x, c), & j = 1, \dots, n, \\ f_{u_j^x}(\beta) &= z(l^c, u^c, l^x, u^x + \beta e_j, c), & j = 1, \dots, n, \\ f_{c_j}(\beta) &= z(l^c, u^c, l^x, u^x, c + \beta e_j), & j = 1, \dots, n. \end{aligned} \quad (11.7)$$

Given these definitions it should be clear how linearity intervals and shadow prices are defined for the parameters u_i^c etc.

11.4.1.1 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

$$f_{e_i^c}(\beta) = z(l^c + \beta e_i, u^c + \beta e_i, l^x, u^x, c) \quad (11.8)$$

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.

11.4.2 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.

11.4.3 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 (11.1), i.e. x^* and $((s_l^c)^*, (s_u^c)^*, (s_l^x)^*, (s_u^x)^*)$ the optimal objective value is given by

$$z^* := c^T x^*. \quad (11.9)$$

The left and right shadow prices σ_1 and σ_2 for l_i^c are given by this pair of optimization problems:

$$\begin{aligned} \sigma_1 &= \text{minimize} && e_i^T s_l^c \\ &\text{subject to} && A^T(s_l^c - s_u^c) + s_l^x - s_u^x = c, \\ &&& (l_c)^T(s_l^c) - (u_c)^T(s_u^c) + (l_x)^T(s_l^x) - (u_x)^T(s_u^x) = z^*, \\ &&& s_l^c, s_u^c, s_l^x, s_u^x \geq 0 \end{aligned} \quad (11.10)$$

and

$$\begin{aligned} \sigma_2 &= \text{maximize} && e_i^T s_l^c \\ &\text{subject to} && A^T(s_l^c - s_u^c) + s_l^x - s_u^x = c, \\ &&& (l_c)^T(s_l^c) - (u_c)^T(s_u^c) + (l_x)^T(s_l^x) - (u_x)^T(s_u^x) = z^*, \\ &&& s_l^c, s_u^c, s_l^x, s_u^x \geq 0. \end{aligned} \quad (11.11)$$

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

$$(s_l^c)^*_i \in [\sigma_1, \sigma_2]. \quad (11.12)$$

Next, the linearity interval $[\beta_1, \beta_2]$ for l_i^c is computed by solving the two optimization problems

$$\begin{aligned} \beta_1 &= \text{minimize} && \beta \\ &\text{subject to} && l^c + \beta e_i \leq Ax \leq u^c, \\ &&& c^T x - \sigma_1 \beta = z^*, \\ &&& l^x \leq x \leq u^x, \end{aligned} \quad (11.13)$$

and

$$\begin{aligned} \beta_2 &= \text{maximize} && \beta \\ &\text{subject to} && l^c + \beta e_i \leq Ax \leq u^c, \\ &&& c^T x - \sigma_2 \beta = z^*, \\ &&& l^x \leq x \leq u^x. \end{aligned} \quad (11.14)$$

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:

$$\begin{aligned} \sigma_1 &= \text{minimize} && e_j^T x \\ &\text{subject to} && l^c + \beta e_i \leq Ax \leq u^c, \\ &&& c^T x = z^*, \\ &&& l^x \leq x \leq u^x \end{aligned} \quad (11.15)$$

and

$$\begin{aligned} \sigma_2 = \text{maximize} \quad & e_j^T x \\ \text{subject to} \quad & l^c + \beta e_i \leq Ax \leq u^c, \\ & c^T x = z^*, \\ & l^x \leq x \leq u^x. \end{aligned} \quad (11.16)$$

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

$$x_j^* \in [\sigma_1, \sigma_2]. \quad (11.17)$$

The linearity interval $[\beta_1, \beta_2]$ for a c_j is computed as follows:

$$\begin{aligned} \beta_1 = \text{minimize} \quad & \beta \\ \text{subject to} \quad & A^T(s_l^c - s_u^c) + s_l^x - s_u^x = c + \beta e_j, \\ & (l_c)^T(s_l^c) - (u_c)^T(s_u^c) + (l_x)^T(s_l^x) - (u_x)^T(s_u^x) - \sigma_1 \beta \leq z^*, \\ & s_l^c, s_u^c, s_l^x, s_u^x \geq 0 \end{aligned} \quad (11.18)$$

and

$$\begin{aligned} \beta_2 = \text{maximize} \quad & \beta \\ \text{subject to} \quad & A^T(s_l^c - s_u^c) + s_l^x - s_u^x = c + \beta e_j, \\ & (l_c)^T(s_l^c) - (u_c)^T(s_u^c) + (l_x)^T(s_l^x) - (u_x)^T(s_u^x) - \sigma_2 \beta \leq z^*, \\ & s_l^c, s_u^c, s_l^x, s_u^x \geq 0. \end{aligned} \quad (11.19)$$

11.4.4 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 11.2.

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

$$\begin{aligned} \text{minimize} \quad & 1x_{11} + 2x_{12} + 5x_{23} + 2x_{24} + 1x_{31} + 2x_{33} + 1x_{34} \end{aligned} \quad (11.20)$$

subject to

$$\begin{aligned} x_{11} + x_{12} & \leq 400, \\ x_{23} + x_{24} & \leq 1200, \\ x_{31} + x_{33} + x_{34} & \leq 1000, \\ x_{11} + x_{31} & = 800, \\ x_{12} & = 100, \\ x_{23} + x_{33} & = 500, \\ x_{24} + x_{34} & = 500, \\ x_{11}, x_{12}, x_{23}, x_{24}, x_{31}, x_{33}, x_{34} & \geq 0. \end{aligned} \quad (11.21)$$

The basis type and the optimal partition type sensitivity results for the transportation problem are shown in Table 11.1 and 11.2 respectively.

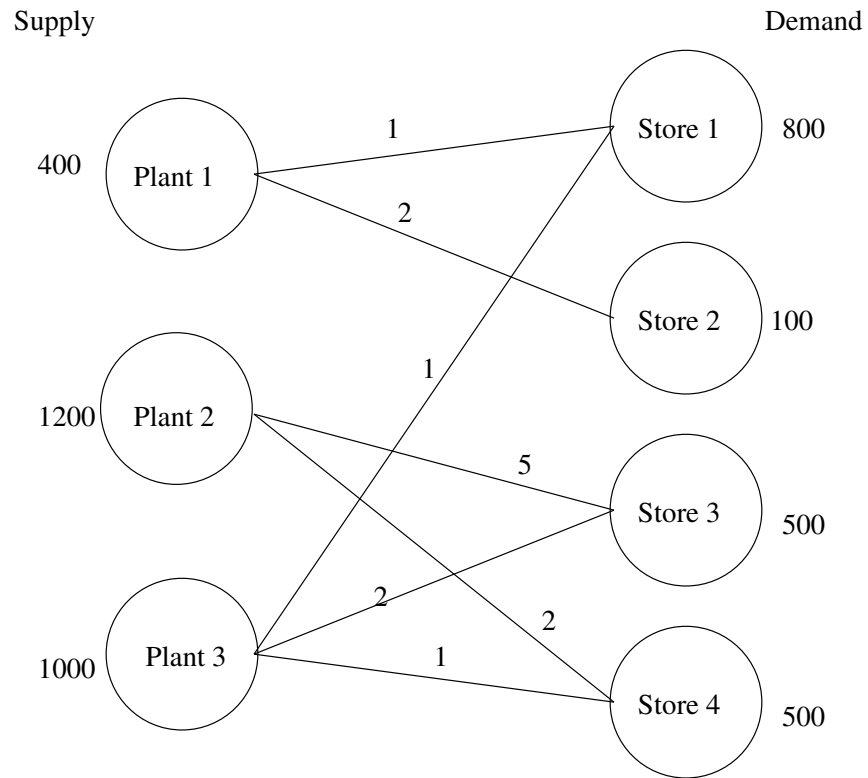


Figure 11.2: Supply, demand and cost of transportation.

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 11.1: 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 11.2: 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

$$\beta \in [0, \beta_1] = [0, 500] \quad (11.22)$$

then the optimal objective value will decrease by the value

$$\sigma_2 \beta = 1\beta. \quad (11.23)$$

Correspondingly, if the upper bound on constraint 1 is decreased by

$$\beta \in [0, 300] \quad (11.24)$$

then the optimal objective value will increase by the value

$$\sigma_1 \beta = 3\beta. \quad (11.25)$$

11.5 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/5/tools/examp/sensitivity.py

```
##
# Copyright: Copyright (c) 1998-2007 MOSEK ApS, Denmark. All rights reserved.
#
# File:      sensitivity.cs
#
# Purpose:   To demonstrate how to perform sensitivity
# analysis from the API on a small problem:
#
# minimize
#
# obj: +1 x11 + 2 x12 + 5 x23 + 2 x24 + 1 x31 + 2 x33 + 1 x34
# st
# c1:   + x11 + x12                                     <= 400
# c2:           + x23 + x24                             <= 1200
# c3:           + x31 + x33 + x34                       <= 1000
# c4:   + x11                                     + x31   = 800
# c5:           + x12                                     = 100
# c6:           + x23                                     + x33   = 500
# c7:           + x24                                     + x34   = 500
#
# The example uses basis type sensitivity analysis.
##
```

```

import sys

import pymosek as mosek

try:
    from Numeric import array, Float, zeros, ones
except ImportError:
    from mosekarr import array, Float, zeros, ones

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

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

# We might write everything directly as a script, but it looks nicer
# to create a function.
def main():
    # Create a handle to MOSEK
    mskhandle = mosek.mosek()
    # Create a MOSEK environment
    env = mskhandle.Env()
    # Attach a printer to the environment
    env.set_Stream(mosek.streamtype.log, streamprinter)
    # Initialize the environment
    env.init()

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

    # Set up data

    bkc = array([ mosek.boundkey.up, mosek.boundkey.up,
                  mosek.boundkey.up, mosek.boundkey.fx,
                  mosek.boundkey.fx, mosek.boundkey.fx,
                  mosek.boundkey.fx ])
    blc = array([ -inf, -inf, -inf, 800., 100., 500., 500. ])
    buc = array([ 400., 1200., 1000., 800., 100., 500., 500. ])

    bkc = array([ mosek.boundkey.lo, mosek.boundkey.lo,
                  mosek.boundkey.lo, mosek.boundkey.lo,
                  mosek.boundkey.lo, mosek.boundkey.lo,
                  mosek.boundkey.lo ])
    c = array([ 1.0, 2.0, 5.0, 2.0, 1.0, 2.0, 1.0 ])
    blx = array([ 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0 ])
    bux = array([ inf, inf, inf, inf, inf, inf, inf ])

    ptrb = array([ 0, 2, 4, 6, 8, 10, 12 ])

```

```

ptre = array([ 2,4,6,8,10,12,14 ])
sub  = array([ 0,3,0,4,1,5,1,6,2,3,2,5,2,6 ])

val  = array([ 1.0,1.0,1.0,1.0,1.0,1.0,1.0,
              1.0,1.0,1.0,1.0,1.0,1.0,1.0 ])

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

# Input linear data
task.inputdata(numcon,numvar,
               c,0.0,
               ptrb, ptre, sub, val,
               bkc, blc, buc,
               bkx, blx, bux)

# Set objective sense
task.putobjsense(mosek.objsense.minimize)

# Optimize
task.optimize();

# Analyze upper bound on c1 and the equality constraint on c4
subi  = array([ 0, 3 ])
marki = array([ mosek.mark.up, mosek.mark.up ])

# Analyze lower bound on the variables x12 and x31
subj  = array([ 1, 4 ])
markj = array([ mosek.mark.lo, mosek.mark.lo ])

leftpricei = zeros(2,Float)
rightpricei = zeros(2,Float)
leftrangei = zeros(2,Float)
rightrangei = zeros(2,Float)
leftpricej = zeros(2,Float)
rightpricej = zeros(2,Float)
leftrangej = zeros(2,Float)
rightrangej = zeros(2,Float)

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

print 'Results from sensitivity analysis on bounds:'
print '\tleftprice | rightprice | leftrange | rightrange '
print 'For constraints:'

for i in range(2):

```



```

        print '\t%10f    %10f    %10f    %10f' % (leftpricei[i],
                                                    rightpricei[i],
                                                    leftrangei[i],
                                                    rightrangei[i])

    print 'For variables:'
    for i in range(2):
        print '\t%10f    %10f    %10f    %10f' % (leftpricej[i],
                                                    rightpricej[i],
                                                    leftrangej[i],
                                                    rightrangej[i])

    leftprice  = zeros(2,Float)
    rightprice = zeros(2,Float)
    leftrange  = zeros(2,Float)
    rightrange = zeros(2,Float)
    subc       = array([ 2, 5 ])

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

    print 'Results from sensitivity analysis on objective coefficients:'

    for i in range(2):
        print '\t%10f    %10f    %10f    %10f' % (leftprice[i],
                                                    rightprice[i],
                                                    leftrange[i],
                                                    rightrange[i])

    return None

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

```

11.6 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.

11.6.1 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|UL [cname1]
  U|L|UL [cname2]-[cname3]
BOUNDS VARIABLES
  U|L|UL [vname1]
  U|L|UL [vname2]-[vname3]
OBJECTIVE VARIABLES
  [vname1]
  [vname2]-[vname3]
```

Figure 11.3: The sensitivity analysis file format.

The format of the sensitivity specification file is shown in figure 11.3, 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 11.4.1.1 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.

11.6.2 Example: Sensitivity analysis from command line

As an example consider the `sensitivity.ssp` file shown in Figure 11.4.

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 11.4: 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	

11.6.3 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 12

API developer guidelines

The purpose of this chapter is to present some guidelines for developing an application which uses the MOSEK API.

12.1 Turn on logging

While developing a new application it is beneficial to turn on logging so that error and diagnostics messages are displayed.

See the explained example in section 5.2 for instructions on turning log output on. You should also always catch and handle any exceptions thrown by MOSEK.

More log information can be obtained by modifying one or more of the parameters:

- `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
```

12.2 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 numbers and outputs a warning if any are found.

12.3 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:

```
task.Task.writedata("taskdump.opf")
task.Task.optimize()
```

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.

12.4 Error handling

12.5 Check the problem status and solution status

If a problem is primal or dual infeasible and MOSEK detects this, it is **not** reported as an error. Therefore, it is important to check the problem status and solution status after the optimization optimization ended using the `Task.getsolutionstatus` function or the `Task.getsolutioninf` function.

12.6 Important API limitations

12.6.1 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.

12.7 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 13

API reference

This chapter lists all functionality in the MOSEK Python API.

13.1 API Functionality

Functions in the interface grouped by functionality.

13.1.1 Reading and writing data files.

Reading and writing data files.

mosek.Task.readbranchpriorities (page 207)

Reads branching priority data from a file.

mosek.Task.readdata (page 207)

Reads problem data from a file.

mosek.Task.readparamfile (page 208)

Reads a parameter file.

mosek.Task.readsolution (page 208)

Reads a solution from a file.

mosek.Task.readsummary (page 208)

Prints information about last file read.

mosek.Task.writebranchpriorities (page 215)

Writes branching priority data to a file.

mosek.Task.writeparamfile (page 216)

Writes all the parameters to a parameter file.

mosek.Task.writesolution (page 216)
Write a solution to a file.

13.1.2 Solutions.

Obtain or define a solution.

mosek.Task.deletesolution (page 162)
Undefineds a solution and frees the memory it uses.

mosek.Task.getdualobj (page 170)
Obtains the dual objective value.

mosek.Task.getprimalobj (page 175)
Obtains the primal objective value.

mosek.Task.getreducedcosts (page 177)
Obtains the difference of slx-sux for a sequence of variables.

mosek.Task.getsolution (page 177)
Obtains the complete solution.

mosek.Task.getsolutioni (page 179)
Obtains the solution for a single constraint or variable.

mosek.Task.getsolutioninf (page 180)
Obtains information about a solution.

mosek.Task.getsolutionslice (page 180)
Obtains a slice of the solution.

mosek.Task.getsolutionstatus (page 182)
Obtains information about the problem and solution statuses.

mosek.Task.getsolutionstatuskeyslice (page 182)
Obtains a slice of the solution status keys.

mosek.Task.makesolutionstatusunknown (page 187)
Sets the solution status to unknown.

mosek.Task.putsolution (page 204)
Inserts a solution.

mosek.Task.putsolutioni (page 205)
Sets the primal and dual solution information for a single constraint or variable.

mosek.Task.readsolution (page 208)
Reads a solution from a file.

mosek.Task.solutiondef (page 213)
Checks whether a solution is defined.

mosek.Task.solutionsummary (page 213)
Prints a short summary of a solution.

mosek.Task.undefsolution (page 215)
Undefines a solution.

mosek.Task.writedata (page 215)
Writes problem data to a file.

13.1.3 Memory allocation and deallocation.

Memory allocation and deallocation.

mosek.Task.checkmemtask (page 161)
Checks the memory allocated by the task.

mosek.Task.getmemusagetask (page 172)
Obtains information about the amount of memory used by a task.

13.1.4 Changing problem specification.

Input or change problem specification

mosek.Task.append (page 158)
Appends a number of variables or constraints to the optimization task.

mosek.Task.appendcone (page 158)
Appends a new cone constraint to the problem.

mosek.Task.appendcons (page 159)
Appends one or more constraints and specifies bounds and A coefficients.

mosek.Task.appendvars (page 160)
Appends one or more variables and specifies bounds on variables, c coefficients and A coefficients.

mosek.Task.chgbound (page 161)
Changes the bounds for one constraint or variable.

mosek.Task.clonetask (page 162)
Creates a clone of an existing task.

mosek.Task.commitchanges (page 162)
Commits all cached problem changes.

mosek.Task.inputdata (page 185)
Input the linear part of an optimization task in one function call.

mosek.Task.putaij (page 191)
Changes a single value in the linear coefficient matrix.

- mosek.Task.putaijlist** (page 192)
Changes one or more coefficients in A .
- mosek.Task.putavec** (page 192)
Replaces all elements in one row or column of A .
- mosek.Task.putaveclist** (page 193)
Replaces all elements in one or more rows or columns in A by new values.
- mosek.Task.putbound** (page 194)
Changes the bound for either one constraint or one variable.
- mosek.Task.putboundlist** (page 195)
Changes the bounds of constraints or variables.
- mosek.Task.putboundslice** (page 195)
Modifies bounds.
- mosek.Task.putcfix** (page 196)
Replaces the fixed term in the objective.
- mosek.Task.putcj** (page 196)
Modifies one linear coefficient in the objective.
- mosek.Task.putclist** (page 196)
Modifies a part of c .
- mosek.Task.putcone** (page 197)
Replaces a conic constraint.
- mosek.Task.putobjsense** (page 200)
Sets the objective sense.
- mosek.Task.putqcon** (page 201)
Replaces all quadratic terms in constraints.
- mosek.Task.putqconk** (page 202)
Replaces all quadratic terms in a single constraint.
- mosek.Task.putqobj** (page 203)
Replaces all quadratic terms in the objective.
- mosek.Task.putqobjij** (page 203)
Replaces one of the coefficients in the quadratic term in the objective.
- mosek.Task.putvartype** (page 206)
Sets the variable type of one variable.
- mosek.Task.putvartypelist** (page 207)
Sets the variable type for one or more variables.

13.1.5 Delete problem elements (variables,constraints,cones).

Functionality for deleting problem elements such as variables, constraints or cones.

mosek.Task.remove (page 211)

The function removes a number of constraints or variables.

mosek.Task.removecone (page 211)

Removes a conic constraint from the problem.

13.1.6 Add problem elements (variables,constraints,cones).

Functionality for adding problem elements such as variables, constraints or cones.

mosek.Task.append (page 158)

Appends a number of variables or constraints to the optimization task.

mosek.Task.appendcone (page 158)

Appends a new cone constraint to the problem.

13.1.7 Inspect problem specification.

Functionality for inspecting the problem specification (A, Q , bounds, objective e.t.c).

mosek.Task.getaij (page 163)

Obtains a single coefficient in A .

mosek.Task.getaslice (page 164)

Obtains a sequence of rows or columns from A .

mosek.Task.getaslicetrip (page 165)

Obtains a sequence of rows or columns from A in triplet format.

mosek.Task.getavec (page 166)

Obtains one row or column of A .

mosek.Task.getavecnumnz (page 166)

Obtains the number of non-zero elements in one row or column of A .

mosek.Task.getbound (page 167)

Obtains bound information for one constraint or variable.

mosek.Task.getboundslice (page 167)

Obtains bounds information for a sequence of variables or constraints.

mosek.Task.getc (page 168)

Obtains all objective coefficients c .

- mosek.Task.getcfix** (page 168)
Obtains the fixed term in the objective.
- mosek.Task.getcone** (page 168)
Obtains a conic constraint.
- mosek.Task.getconeinfo** (page 168)
Obtains information about a conic constraint.
- mosek.Task.getcslice** (page 169)
Obtains a part of c .
- mosek.Task.getnumanz** (page 173)
Obtains the number of non-zeros in A .
- mosek.Task.getnumcon** (page 173)
Obtains the number of constraints.
- mosek.Task.getnumcone** (page 173)
Obtains the number of cones.
- mosek.Task.getnumconemem** (page 174)
Obtains the number of members in a cone.
- mosek.Task.getnumintvar** (page 174)
Obtains the number of integer constrained variables.
- mosek.Task.getnumqconnz** (page 174)
Obtains the number of non-zero quadratic terms in a constraint.
- mosek.Task.getnumqobjnz** (page 175)
Obtains the number of non-zero quadratic terms in the objective.
- mosek.Task.getnumvar** (page 175)
Obtains the number of variables.
- mosek.Task.getobjsense** (page 175)
Gets the objective sense.
- mosek.Task.getprobtype** (page 176)
Obtains the problem type.
- mosek.Task.getqconk** (page 176)
Obtains all the quadratic terms in a constraint.
- mosek.Task.getqobj** (page 176)
Obtains all the quadratic terms in the objective.
- mosek.Task.getqobjij** (page 177)
Obtains one coefficient from the quadratic term of the objective
- mosek.Task.getvartype** (page 184)
Gets the variable type of one variable.

mosek.Task.getvartypelist (page 184)
Obtains the variable type for one or more variables.

13.1.8 Conic constraints.

Functionality related to conic terms in the problem.

mosek.Task.appendcone (page 158)
Appends a new cone constraint to the problem.

mosek.Task.getcone (page 168)
Obtains a conic constraint.

mosek.Task.getconeinfo (page 168)
Obtains information about a conic constraint.

mosek.Task.getnumcone (page 173)
Obtains the number of cones.

mosek.Task.putcone (page 197)
Replaces a conic constraint.

mosek.Task.removecone (page 211)
Removes a conic constraint from the problem.

13.1.9 Bounds.

Functionality related to changing or inspecting bounds on variables or constraints.

mosek.Task.chgbound (page 161)
Changes the bounds for one constraint or variable.

mosek.Task.getbound (page 167)
Obtains bound information for one constraint or variable.

mosek.Task.getboundslice (page 167)
Obtains bounds information for a sequence of variables or constraints.

mosek.Task.putbound (page 194)
Changes the bound for either one constraint or one variable.

mosek.Task.putboundlist (page 195)
Changes the bounds of constraints or variables.

mosek.Task.putboundslice (page 195)
Modifies bounds.

13.1.10 Output stream functions.

Output stream functions.

- mosek.Env.echointro** (page 145)
Prints an intro to message stream.
- mosek.Env.linkfiletoenvstream** (page 146)
Directs all output from a stream to a file.
- mosek.Task.linkfiletotaskstream** (page 187)
Directs all output from a task stream to a file.
- mosek.Task.printdata** (page 190)
Prints a part of the problem data to a stream.
- mosek.Task.printparam** (page 191)
Prints the current parameter settings.
- mosek.Task.readsummary** (page 208)
Prints information about last file read.
- mosek.Task.solutionsummary** (page 213)
Prints a short summary of a solution.

13.1.11 Objective function.

Change or inspect objective function.

- mosek.Task.getc** (page 168)
Obtains all objective coefficients c .
- mosek.Task.getcfix** (page 168)
Obtains the fixed term in the objective.
- mosek.Task.getcslice** (page 169)
Obtains a part of c .
- mosek.Task.getdualobj** (page 170)
Obtains the dual objective value.
- mosek.Task.getnumqobjnz** (page 175)
Obtains the number of non-zero quadratic terms in the objective.
- mosek.Task.getobjname** (page 175)
Obtains the name assigned to the objective function.
- mosek.Task.getobjsense** (page 175)
Gets the objective sense.

mosek.Task.getprimalobj (page 175)

Obtains the primal objective value.

mosek.Task.getqobj (page 176)

Obtains all the quadratic terms in the objective.

mosek.Task.getqobjjj (page 177)

Obtains one coefficient from the quadratic term of the objective

mosek.Task.putcfix (page 196)

Replaces the fixed term in the objective.

mosek.Task.putcj (page 196)

Modifies one linear coefficient in the objective.

mosek.Task.putclist (page 196)

Modifies a part of c .

mosek.Task.putobjsense (page 200)

Sets the objective sense.

mosek.Task.putqobj (page 203)

Replaces all quadratic terms in the objective.

mosek.Task.putqobjjj (page 203)

Replaces one of the coefficients in the quadratic term in the objective.

13.1.12 Inspect statistics from the optimizer.

Inspect statistics from the optimizer.

mosek.Task.appendstat (page 160)

Appends a record the statistics file.

mosek.Task.getdouinf (page 169)

Obtains a double information item.

mosek.Task.getintinf (page 170)

Obtains an integer information item.

mosek.Task.startstat (page 214)

Starts the statistics file.

mosek.Task.stopstat (page 214)

Stops the statistics file.

13.1.13 Parameters (set/get).

Setting and inspecting solver parameters.

mosek.Task.getdouparam (page 170)
Obtains a double parameter.

mosek.Task.getintparam (page 171)
Obtains an integer parameter.

mosek.Task.getnumparam (page 174)
Obtains the number of parameters of a given type.

mosek.Task.getstrparam (page 183)
Obtains the value of a string parameter.

mosek.Env.getsymbcondim (page 145)
Obtains dimensional information for the defined symbolic constants.

mosek.Task.isdouparname (page 186)
Checks a double parameter name.

mosek.Task.isintparname (page 186)
Checks an integer parameter name.

mosek.Task.isstrparname (page 186)
Checks a string parameter name.

mosek.Task.putdouparam (page 197)
Sets a double parameter.

mosek.Task.putintparam (page 197)
Sets an integer parameter.

mosek.Task.putnadouparam (page 199)
Sets a double parameter.

mosek.Task.putnaintparam (page 199)
Sets an integer parameter.

mosek.Task.putnastrparam (page 200)
Sets a string parameter.

mosek.Task.putparam (page 201)
Modifies the value of parameter.

mosek.Task.putstrparam (page 206)
Sets a string parameter.

mosek.Task.setdefaults (page 212)
Resets all parameters values.

13.1.14 Naming.

Functionality related to naming.

mosek.Task.getconname (page 169)
Obtains a name of a constraint.

mosek.Task.getmaxnamelen (page 171)
Obtains the maximum length of any objective, constraint, variable or cone name.

mosek.Task.getname (page 172)
Obtains the name of a cone, a variable or a constraint.

mosek.Task.getnameindex (page 173)
Checks whether a name has been assigned and returns the index corresponding to the name.

mosek.Task.getobjname (page 175)
Obtains the name assigned to the objective function.

mosek.Task.gettaskname (page 183)
Obtains the task name.

mosek.Task.getvarname (page 184)
Obtains a name of a variable.

mosek.Task.putname (page 200)
Assigns the name **name** to a problem item such as a constraint.

mosek.Task.putobjname (page 200)
Assigns a new name to the objective.

mosek.Task.puttaskname (page 206)
Assigns a new name to the task.

13.1.15 Preallocating space for problem data.

Functionality related to preallocating space for problem data.

mosek.Task.getmaxnumanz (page 171)
Obtains number of preallocated non-zeros for A .

mosek.Task.getmaxnumcon (page 171)
Obtains the number of preallocated constraints in the optimization task.

mosek.Task.getmaxnumcone (page 171)
Obtains the number of preallocated cones in the optimization task.

mosek.Task.getmaxnumqnz (page 172)
Obtains the number of preallocated non-zeros for Q (both objective and constraints).

mosek.Task.getmaxnumvar (page 172)

Obtains the maximum number variables allowed.

mosek.Task.putmaxnumanz (page 197)

The function changes the size of the preallocated storage for linear coefficients.

mosek.Task.putmaxnumcon (page 198)

Sets the number of preallocated constraints in the optimization task.

mosek.Task.putmaxnumcone (page 198)

Sets the number of preallocated conic constraints in the optimization task.

mosek.Task.putmaxnumqnz (page 199)

Changes the size of the preallocated storage for Q .

mosek.Task.putmaxnumvar (page 199)

Sets the number of preallocated variables in the optimization task.

13.1.16 Integer variables.

Functionality related to integer variables.

mosek.Task.getnumintvar (page 174)

Obtains the number of integer constrained variables.

mosek.Task.getvarbranchdir (page 183)

Obtains the branching direction for a variable.

mosek.Task.getvarbranchpri (page 183)

Obtains the branching priority for a variable.

mosek.Task.getvartype (page 184)

Gets the variable type of one variable.

mosek.Task.getvartypelist (page 184)

Obtains the variable type for one or more variables.

mosek.Task.putvarbranchorder (page 206)

Assigns a branching priority and direction to a variable.

mosek.Task.putvartype (page 206)

Sets the variable type of one variable.

mosek.Task.putvartypelist (page 207)

Sets the variable type for one or more variables.

13.1.17 Quadratic terms.

Functionality related to quadratic terms.

- mosek.Task.getqconk** (page 176)
Obtains all the quadratic terms in a constraint.
- mosek.Task.getqobj** (page 176)
Obtains all the quadratic terms in the objective.
- mosek.Task.getqobjjj** (page 177)
Obtains one coefficient from the quadratic term of the objective
- mosek.Task.putqcon** (page 201)
Replaces all quadratic terms in constraints.
- mosek.Task.putqconk** (page 202)
Replaces all quadratic terms in a single constraint.
- mosek.Task.putqobj** (page 203)
Replaces all quadratic terms in the objective.
- mosek.Task.putqobjjj** (page 203)
Replaces one of the coefficients in the quadratic term in the objective.

13.1.18 Diagnosing infeasibility.

Functions for diagnosing infeasibility.

- mosek.Task.getinfeasiblesubproblem** (page 170)
Obtains an infeasible sub problem.
- mosek.Task.relaxprimal** (page 208)
Creates a problem that finds the minimal change to the bounds that makes an infeasible problem feasible.

13.1.19 Optimization.

Functions for optimization.

- mosek.Task.checkdata** (page 161)
Checks data of the task.
- mosek.Task.optimize** (page 187)
Optimizes the problem.
- mosek.Task.optimizeconcurrent** (page 187)
Optimize a given task with several optimizers concurrently.

mosek.Task.optimize (page 188)
Optimizes the problem.

13.1.20 Sensitivity analysis.

Functions for sensitivity analysis.

mosek.Task.dualsensitivity (page 163)
Performs sensitivity analysis on objective coefficients.

mosek.Task.primalsensitivity (page 188)
Perform sensitivity analysis on bounds.

mosek.Task.sensitivityreport (page 212)
Creates a sensitivity report.

13.1.21 Testing data validity.

Functions for testing data validity.

mosek.Task.checkconvexity (page 161)
Checks if a quadratic optimization problem is convex.

13.1.22 Solving with the basis.

Functions for solving linear systems with the basis matrix.

mosek.Task.initbasissolve (page 184)
Prepare a task for use with the **Task.solvewithbasis** function.

mosek.Task.solvewithbasis (page 213)
Solve a linear equation system involving a basis matrix.

13.1.23 Initialization of environment.

Creation and initialization of environment.

mosek.Env.initenv (page 145)
Initialize a MOSEK environment.

mosek.Env.putlicensedefaults (page 147)
Set defaults used by the license manager.

13.1.24 Change A .

Change elements in the coefficient (A) matrix.

mosek.Task.appendcons (page 159)

Appends one or more constraints and specifies bounds and A coefficients.

mosek.Task.appendvars (page 160)

Appends one or more variables and specifies bounds on variables, c coefficients and A coefficients.

mosek.Task.commitchanges (page 162)

Commits all cached problem changes.

mosek.Task.putaij (page 191)

Changes a single value in the linear coefficient matrix.

mosek.Task.putaijlist (page 192)

Changes one or more coefficients in A .

mosek.Task.putavec (page 192)

Replaces all elements in one row or column of A .

mosek.Task.putaveclist (page 193)

Replaces all elements in one or more rows or columns in A by new values.

13.2 Class mosek.ArrayLengthException

Derived from:

`__builtin___.Exception`

Description:

This exception is raised if an input or output array was shorter than required.

13.3 Class mosek.Env

Description:

A Mosek Environment

13.3.1 Constructors

- `mosek.Env`

Syntax:

`Env ()`

Description:

Create a MOSEK environment object.

- `mosek.Env`

Syntax:

`Env (str dbgfile)`

Description:

Create a MOSEK environment object.

Arguments:

`dbgfile` A file which will be used to log memory debugging information from MOSEK

13.3.2 Attributes

- `mosek.CtrlC` `CtrlC` (write only) Control-c call-back object.
- `mosek.Exit` `Exit` (write only) Exit handler call-back object.

13.3.3 Methods

- `mosek.Env.echointro` 145
Prints an intro to message stream.
- `mosek.Env.getsymbcondim` 145
Obtains dimensional information for the defined symbolic constants.
- `mosek.Env.getversion` 145
Obtains MOSEK version information.
- `mosek.Env.initenv` 145
Initialize a MOSEK environment.
- `mosek.Env.isinfinity` 146
Return true if `value` considered infinity by MOSEK.
- `mosek.Env.linkfiletoenvstream` 146
Directs all output from a stream to a file.
- `mosek.Env.putcpudefaults` 146
Set defaults default CPU type and cache sizes.
- `mosek.Env.putdllpath` 146
Sets the path to the DLL/shared libraries that MOSEK is loading.
- `mosek.Env.putkeepdlls` 147
Controls whether explicitly loaded DLLs should be kept.

- `mosek.Env.putlicensedefaults` 147
Set defaults used by the license manager.

- `mosek.Env.set_Stream` 147
Attach a stream call-back handler.

- `mosek.Env.echointro`

Syntax:

```
echointro (int longver)
```

Arguments:

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

Description: Prints an intro to message stream.

- `mosek.Env.getsymbcondim`

Syntax:

```
getsymbcondim (
    List|None num,
    List|None maxlen)
```

Arguments:

`num` (**output**) Number of symbolic constants defined by MOSEK.

`maxlen` (**output**) Maximum length of the name of any symbolic constants.

Description: Obtains the number of symbolic constants defined by MOSEK and the maximum length of the name of any symbolic constant.

- `mosek.Env.getversion`

Syntax:

```
getversion (
    List|None major,
    List|None minor,
    List|None build,
    List|None revision)
```

Arguments:

`major` (**output**) Major version number. Modified only if a non-null pointer.

`minor` (**output**) Minor version number. Modified only if a non-null pointer.

`build` (**output**) Build number. Modified only if a non-null pointer.

`revision` (**output**) Revision number. Modified only if a non-null pointer.

Description: Obtains MOSEK version information.

- `mosek.Env.initenv`

Syntax:

```
initenv ()
```

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.isinfinity`

Syntax:

```
isinfinity (double value)
```

Arguments:

value

Description: Return true if value considered infinity by MOSEK

- `mosek.Env.linkfiletoenvstream`

Syntax:

```
linkfiletoenvstream (
    mosek.streamtype whichstream,
    str filename,
    int append)
```

Arguments:

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)
```

Arguments:

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)
```

Arguments:

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)
```

Arguments:

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)
```

Arguments:

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 used as the default value for

`mosek.iparam.license_pause_time`

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)
```

Arguments:

whichstream Index of the stream.

Description: Attach a stream call-back handler.

13.4 Class `mosek.Error`

Derived from:

`mosek.Exception`

Description:

This is an exception class representing MOSEK errors.

13.4.1 Constructors

- `mosek.Error`

Syntax:

```
Error (mosek.rescode code)
```

Description:

Construct an error from a MOSEK error code.

Arguments:

code The MOSEK response code to create the exception from.

- `mosek.Error`

Syntax:

```
Error (
    mosek.rescode code,
    str msg)
```

Description:

Construct an error from a MOSEK error code and a message.

Arguments:

code The MOSEK response code to create the exception from.

msg A message describing the error situation.

13.5 Class mosek.Exception

Derived from:

`__builtin__.Exception`

Description:

This is the base class for exceptions based on MOSEK response codes.

13.5.1 Constructors

- `mosek.Exception`

Syntax:

```
Exception (mosek.rescode code)
```

Description:

Construct an exception from a MOSEK error code.

Arguments:

`code` The MOSEK response code to create the exception from.

13.6 Class mosek.Task

Description:

A Mosek Optimization task

13.6.1 Constructors

- `mosek.Task`

Syntax:

```
Task (
    int maxnumcon,
    int maxnumvar)
```

Description:

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.

Arguments:

`maxnumcon` Initially reserve space for this many constraints.

`maxnumvar` Initially reserve space for this many variables.

13.6.2 Attributes

- `mosek.ErrorHandler ErrorHandler` (write only) Error handler call-back object
- `mosek.Nonlinear Nonlinear` (write only) Nonlinear sparsity and value computation call-back object.
- `mosek.Progress Progress` (write only) Progress call-back object

13.6.3 Methods

- `mosek.Task.append` 158
Appends a number of variables or constraints to the optimization task.
- `mosek.Task.appendcone` 158
Appends a new cone constraint to the problem.
- `mosek.Task.appendcons` 159
Appends one or more constraints and specifies bounds and A coefficients.
- `mosek.Task.appendstat` 160
Appends a record the statistics file.
- `mosek.Task.appendvars` 160
Appends one or more variables and specifies bounds on variables, c coefficients and A coefficients.
- `mosek.Task.checkconvexity` 161
Checks if a quadratic optimization problem is convex.
- `mosek.Task.checkdata` 161
Checks data of the task.
- `mosek.Task.checkmemtask` 161
Checks the memory allocated by the task.
- `mosek.Task.chgbound` 161
Changes the bounds for one constraint or variable.
- `mosek.Task.clonetask` 162
Creates a clone of an existing task.
- `mosek.Task.commitchanges` 162
Commits all cached problem changes.
- `mosek.Task.deletesolution` 162
Undefineds a solution and frees the memory it uses.
- `mosek.Task.dualsensitivity` 163
Performs sensitivity analysis on objective coefficients.

- `mosek.Task.getaij` 163
Obtains a single coefficient in A .
- `mosek.Task.getapiecenumnz` 164
Obtains the number non-zeros in a rectangular piece of A .
- `mosek.Task.getaslice` 164
Obtains a sequence of rows or columns from A .
- `mosek.Task.getaslicenumnz` 165
Obtains the number of non-zeros in a row or column slice of A .
- `mosek.Task.getaslicetrip` 165
Obtains a sequence of rows or columns from A in triplet format.
- `mosek.Task.getavec` 166
Obtains one row or column of A .
- `mosek.Task.getavecnumnz` 166
Obtains the number of non-zero elements in one row or column of A .
- `mosek.Task.getbound` 167
Obtains bound information for one constraint or variable.
- `mosek.Task.getboundslice` 167
Obtains bounds information for a sequence of variables or constraints.
- `mosek.Task.getc` 168
Obtains all objective coefficients c .
- `mosek.Task.getcfix` 168
Obtains the fixed term in the objective.
- `mosek.Task.getcone` 168
Obtains a conic constraint.
- `mosek.Task.getconeinfo` 168
Obtains information about a conic constraint.
- `mosek.Task.getconname` 169
Obtains a name of a constraint.
- `mosek.Task.getcslice` 169
Obtains a part of c .
- `mosek.Task.getdouinf` 169
Obtains a double information item.
- `mosek.Task.getdoupam` 170
Obtains a double parameter.
- `mosek.Task.getdualobj` 170
Obtains the dual objective value.

• <code>mosek.Task.getinfeasiblesubproblem</code>	170
Obtains an infeasible sub problem.	
• <code>mosek.Task.getintinf</code>	170
Obtains an integer information item.	
• <code>mosek.Task.getintparam</code>	171
Obtains an integer parameter.	
• <code>mosek.Task.getmaxnamelen</code>	171
Obtains the maximum length of any objective, constraint, variable or cone name.	
• <code>mosek.Task.getmaxnumanz</code>	171
Obtains number of preallocated non-zeros for A .	
• <code>mosek.Task.getmaxnumcon</code>	171
Obtains the number of preallocated constraints in the optimization task.	
• <code>mosek.Task.getmaxnumcone</code>	171
Obtains the number of preallocated cones in the optimization task.	
• <code>mosek.Task.getmaxnumqnz</code>	172
Obtains the number of preallocated non-zeros for Q (both objective and constraints).	
• <code>mosek.Task.getmaxnumvar</code>	172
Obtains the maximum number variables allowed.	
• <code>mosek.Task.getmemusagetask</code>	172
Obtains information about the amount of memory used by a task.	
• <code>mosek.Task.getname</code>	172
Obtains the name of a cone, a variable or a constraint.	
• <code>mosek.Task.getnameindex</code>	173
Checks whether a name has been assigned and returns the index corresponding to the name.	
• <code>mosek.Task.getnumanz</code>	173
Obtains the number of non-zeros in A .	
• <code>mosek.Task.getnumcon</code>	173
Obtains the number of constraints.	
• <code>mosek.Task.getnumcone</code>	173
Obtains the number of cones.	
• <code>mosek.Task.getnumconemem</code>	174
Obtains the number of members in a cone.	
• <code>mosek.Task.getnumintvar</code>	174
Obtains the number of integer constrained variables.	
• <code>mosek.Task.getnumparam</code>	174
Obtains the number of parameters of a given type.	

- `mosek.Task.getnumqconnz` 174
Obtains the number of non-zero quadratic terms in a constraint.
- `mosek.Task.getnumqobjnz` 175
Obtains the number of non-zero quadratic terms in the objective.
- `mosek.Task.getnumvar` 175
Obtains the number of variables.
- `mosek.Task.getobjname` 175
Obtains the name assigned to the objective function.
- `mosek.Task.getobjsense` 175
Gets the objective sense.
- `mosek.Task.getprimalobj` 175
Obtains the primal objective value.
- `mosek.Task.getprobtype` 176
Obtains the problem type.
- `mosek.Task.getqconk` 176
Obtains all the quadratic terms in a constraint.
- `mosek.Task.getqobj` 176
Obtains all the quadratic terms in the objective.
- `mosek.Task.getqobjij` 177
Obtains one coefficient from the quadratic term of the objective
- `mosek.Task.getreducedcosts` 177
Obtains the difference of slx-sux for a sequence of variables.
- `mosek.Task.getsolution` 177
Obtains the complete solution.
- `mosek.Task.getsolutioni` 179
Obtains the solution for a single constraint or variable.
- `mosek.Task.getsolutioninf` 180
Obtains information about a solution.
- `mosek.Task.getsolutionslice` 180
Obtains a slice of the solution.
- `mosek.Task.getsolutionstatus` 182
Obtains information about the problem and solution statuses.
- `mosek.Task.getsolutionstatuskeyslice` 182
Obtains a slice of the solution status keys.
- `mosek.Task.getstrparam` 183
Obtains the value of a string parameter.

• <code>mosek.Task.gettaskname</code>	183
Obtains the task name.	
• <code>mosek.Task.getvarbranchdir</code>	183
Obtains the branching direction for a variable.	
• <code>mosek.Task.getvarbranchpri</code>	183
Obtains the branching priority for a variable.	
• <code>mosek.Task.getvarname</code>	184
Obtains a name of a variable.	
• <code>mosek.Task.getvartype</code>	184
Gets the variable type of one variable.	
• <code>mosek.Task.getvartypelist</code>	184
Obtains the variable type for one or more variables.	
• <code>mosek.Task.initbasissolve</code>	184
Prepare a task for use with the <code>Task.solvewithbasis</code> function.	
• <code>mosek.Task.inputdata</code>	185
Input the linear part of an optimization task in one function call.	
• <code>mosek.Task.isdoupname</code>	186
Checks a double parameter name.	
• <code>mosek.Task.isintpname</code>	186
Checks an integer parameter name.	
• <code>mosek.Task.isstrpname</code>	186
Checks a string parameter name.	
• <code>mosek.Task.linkfiletotaskstream</code>	187
Directs all output from a task stream to a file.	
• <code>mosek.Task.makesolutionstatusunknown</code>	187
Sets the solution status to unknown.	
• <code>mosek.Task.optimize</code>	187
Optimizes the problem.	
• <code>mosek.Task.optimizeconcurrent</code>	187
Optimize a given task with several optimizers concurrently.	
• <code>mosek.Task.optimizetrm</code>	188
Optimizes the problem.	
• <code>mosek.Task.primalsensitivity</code>	188
Perform sensitivity analysis on bounds.	
• <code>mosek.Task.printdata</code>	190
Prints a part of the problem data to a stream.	

- `mosek.Task.printparam` 191
Prints the current parameter settings.
- `mosek.Task.putaij` 191
Changes a single value in the linear coefficient matrix.
- `mosek.Task.putaijlist` 192
Changes one or more coefficients in A .
- `mosek.Task.putavec` 192
Replaces all elements in one row or column of A .
- `mosek.Task.putaveclist` 193
Replaces all elements in one or more rows or columns in A by new values.
- `mosek.Task.putbound` 194
Changes the bound for either one constraint or one variable.
- `mosek.Task.putboundlist` 195
Changes the bounds of constraints or variables.
- `mosek.Task.putboundslice` 195
Modifies bounds.
- `mosek.Task.putcfix` 196
Replaces the fixed term in the objective.
- `mosek.Task.putcj` 196
Modifies one linear coefficient in the objective.
- `mosek.Task.putclist` 196
Modifies a part of c .
- `mosek.Task.putcone` 197
Replaces a conic constraint.
- `mosek.Task.putdoupam` 197
Sets a double parameter.
- `mosek.Task.putintparam` 197
Sets an integer parameter.
- `mosek.Task.putmaxnumanz` 197
The function changes the size of the preallocated storage for linear coefficients.
- `mosek.Task.putmaxnumcon` 198
Sets the number of preallocated constraints in the optimization task.
- `mosek.Task.putmaxnumcone` 198
Sets the number of preallocated conic constraints in the optimization task.
- `mosek.Task.putmaxnumqnz` 199
Changes the size of the preallocated storage for Q .

• <code>mosek.Task.putmaxnumvar</code>	199
Sets the number of preallocated variables in the optimization task.	
• <code>mosek.Task.putnadouparam</code>	199
Sets a double parameter.	
• <code>mosek.Task.putnaintparam</code>	199
Sets an integer parameter.	
• <code>mosek.Task.putname</code>	200
Assigns the name <code>name</code> to a problem item such as a constraint.	
• <code>mosek.Task.putnastrparam</code>	200
Sets a string parameter.	
• <code>mosek.Task.putobjname</code>	200
Assigns a new name to the objective.	
• <code>mosek.Task.putobjsense</code>	200
Sets the objective sense.	
• <code>mosek.Task.putparam</code>	201
Modifies the value of parameter.	
• <code>mosek.Task.putqcon</code>	201
Replaces all quadratic terms in constraints.	
• <code>mosek.Task.putqconk</code>	202
Replaces all quadratic terms in a single constraint.	
• <code>mosek.Task.putqobj</code>	203
Replaces all quadratic terms in the objective.	
• <code>mosek.Task.putqobjij</code>	203
Replaces one of the coefficients in the quadratic term in the objective.	
• <code>mosek.Task.putsolution</code>	204
Inserts a solution.	
• <code>mosek.Task.putsolutioni</code>	205
Sets the primal and dual solution information for a single constraint or variable.	
• <code>mosek.Task.putsolutionyi</code>	205
Inputs the dual variable of a solution.	
• <code>mosek.Task.putstrparam</code>	206
Sets a string parameter.	
• <code>mosek.Task.puttaskname</code>	206
Assigns a new name to the task.	
• <code>mosek.Task.putvarbranchorder</code>	206
Assigns a branching priority and direction to a variable.	

• <code>mosek.Task.putvartype</code>	206
Sets the variable type of one variable.	
• <code>mosek.Task.putvartypelist</code>	207
Sets the variable type for one or more variables.	
• <code>mosek.Task.readbranchpriorities</code>	207
Reads branching priority data from a file.	
• <code>mosek.Task.readdata</code>	207
Reads problem data from a file.	
• <code>mosek.Task.readparamfile</code>	208
Reads a parameter file.	
• <code>mosek.Task.readsolution</code>	208
Reads a solution from a file.	
• <code>mosek.Task.readsummary</code>	208
Prints information about last file read.	
• <code>mosek.Task.relaxprimal</code>	208
Creates a problem that finds the minimal change to the bounds that makes an infeasible problem feasible.	
• <code>mosek.Task.remove</code>	211
The function removes a number of constraints or variables.	
• <code>mosek.Task.removecone</code>	211
Removes a conic constraint from the problem.	
• <code>mosek.Task.resizetask</code>	211
Resizes an optimization task.	
• <code>mosek.Task.sensitivityreport</code>	212
Creates a sensitivity report.	
• <code>mosek.Task.set_Stream</code>	212
Attach a stream call-back handler.	
• <code>mosek.Task.setdefaults</code>	212
Resets all parameters values.	
• <code>mosek.Task.solutiondef</code>	213
Checks whether a solution is defined.	
• <code>mosek.Task.solutionsummary</code>	213
Prints a short summary of a solution.	
• <code>mosek.Task.solvewithbasis</code>	213
Solve a linear equation system involving a basis matrix.	

- `mosek.Task.startstat` 214
Starts the statistics file.
- `mosek.Task.stopstat` 214
Stops the statistics file.
- `mosek.Task.strtoconetype` 215
Obtains a cone type code.
- `mosek.Task.strtosk` 215
Obtains a status key.
- `mosek.Task.undefsolution` 215
Undefines a solution.
- `mosek.Task.writebranchpriorities` 215
Writes branching priority data to a file.
- `mosek.Task.writedata` 215
Writes problem data to a file.
- `mosek.Task.writeparamfile` 216
Writes all the parameters to a parameter file.
- `mosek.Task.writesolution` 216
Write a solution to a file.

- `mosek.Task.append`

Syntax:

```
append (
    mosek.accmode accmode,
    int num)
```

Arguments:

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)

```

Arguments:

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

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

to the problem where \mathcal{C} is a convex cone. x^t is a subset of the variables which will be specified by the argument **submem**. 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.appendcons**

Syntax:

```

appendcons (
    array(int) aptrb,
    array(int) aptre,
    array(int) asub,
    array(double) aval,
    array(int) bkc,
    array(double) blc,
    array(double) buc)

```

Arguments:

aptrb (input) See (13.2).
aptre (input) See (13.2).
asub (input) Variable subscripts of the new A coefficients. See (13.2).
aval (input) A coefficients of the new constraints. See (13.2).
bkc (input) Bound keys for constraints to be appended. See (13.1).
blc (input) Lower bounds on constraints to be appended. See (13.1).
buc (input) Upper bounds on constraints to be appended. See (13.1).

Description: The function appends one or more constraints to the optimization task. The bounds and A are modified as follows

$$\begin{aligned}
 l_{\text{numcon}+k}^c &= \text{blc}[k], \quad k = 0, \dots, \text{num} - 1, \\
 u_{\text{numcon}+k}^c &= \text{buc}[k], \quad k = 0, \dots, \text{num} - 1,
 \end{aligned} \tag{13.1}$$

and

$$a_{\text{numcon}+k, \text{asub}[l]} = \text{aval}[l], \quad k = 0, \dots, \text{num} - 1, \quad l = \text{aptrb}[k], \dots, \text{aptre}[k] - 1. \tag{13.2}$$

See also:

`Task.putmaxnumcon` Sets the number of preallocated constraints in the optimization task.

- `mosek.Task.appendstat`

Syntax:

```
appendstat ()
```

Description: Appends a record to the statistics file.

- `mosek.Task.appendvars`

Syntax:

```
appendvars (
    array(double) cval,
    array(int) aptrb,
    array(int) aptre,
    array(int) asub,
    array(double) aval,
    array(int) bxx,
    array(double) blx,
    array(double) bux)
```

Arguments:

cval (input) Values of c for the variables to be appended. See (13.3).

aptrb (input) See (13.4).

aptre (input) See (13.4).

asub (input) Constraint subscripts of the A coefficients to be added. See (13.4).

aval (input) The A coefficients corresponding to the appended variables. See (13.4).

bxx (input) Bound keys on variables to be appended. See (13.3).

blx (input) Lower bounds on variables to be appended. See (13.3).

bux (input) Upper bounds on variables to be appended. See (13.3).

Description: The function appends one or more variables to the optimization problem. Moreover, the function initializes c , A and the bounds corresponding to the appended variables as follows

$$\begin{aligned} c_{\text{numvar}+k} &= \text{cval}[k], & k = 0, \dots, \text{num} - 1, \\ l_{\text{numvar}+k}^x &= \text{blx}[k], & k = 0, \dots, \text{num} - 1, \\ u_{\text{numvar}+k}^x &= \text{bux}[k], & k = 0, \dots, \text{num} - 1, \end{aligned} \tag{13.3}$$

and

$$a_{\text{asub}[l], \text{numvar}+k} = \text{aval}[l], \quad k = 0, \dots, \text{num} - 1, \quad l = \text{aptrb}[k], \dots, \text{aptre}[k] - 1 \tag{13.4}$$

where `numvar` is the number variables before the new variables are appended.

See also:

`Task.putmaxnumvar` Sets the number of preallocated variables in the optimization task.

- `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 returns `mosek.rescode.err_nonconvex` 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.checkmemtask`

Syntax:

```
checkmemtask (
    str file,
    int line)
```

Arguments:

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)
```

Arguments:

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.clonetask`

Syntax:

`Task clonetask ()`

Returns:

Description: Creates a clone of an existing task copying all problem data and parameter settings to a new task. Call-back functions are not copied, so a task containing nonlinear functions cannot be cloned.

- `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)`

Arguments:

whichsol (input) Selects a solution.

Description: Undefined 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)
```

Arguments:

`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 β_1 for the coefficient with index `subj[j]`.

`rightrangej` (**output**) `rightrangej[j]` is the right range β_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 11.5.

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:

```
double getaij (
    int i,
    int j)
```

Arguments:

`i` (**input**) Row index of the coefficient to be returned.

j (input) Column index of the coefficient to be returned.

Returns:

Description: Obtains a single coefficient in A .

- `mosek.Task.getapieceenumnz`

Syntax:

```
int getapieceenumnz (
    int firsti,
    int lasti,
    int firstj,
    int lastj)
```

Arguments:

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.

Returns:

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.getaveccnumnz`.

See also:

`Task.getaveccnumnz` Obtains the number of non-zero elements in one row or column of A .

`Task.getaslicenumnz` Obtains the number of non-zeros in a row or column slice of A .

- `mosek.Task.getaslice`

Syntax:

```
getaslice (
    mosek.accmode accmode,
    int first,
    int last,
    List|None surp,
    array(int) ptrb,
    array(int) ptre,
    array(int) sub,
    array(double) val)
```

Arguments:

accmode (input) Defines whether a column-slice or a row-slice is requested.

first (input) Index of the first row or variable in the sequence.

last (input) Index of the last row or variable plus one 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 numerical elements.

Description: Obtains a sequence of rows or columns from *A* in sparse format.

See also:

Task.getaslicenumnz Obtains the number of non-zeros in a row or column slice of *A*.

- **mosek.Task.getaslicenumnz**

Syntax:

```
int getaslicenumnz (
    mosek.acemode accmode,
    int first,
    int last)
```

Arguments:

accmode (input) Defines whether non-zeros are counted in a column-slice or a row-slice.

first (input) Index of the first row or variable in the sequence.

last (input) Index of the last row or variable plus one in the sequence **plus one**.

Returns:

Description: Obtains the number of non-zeros in a row or column slice of *A*.

- **mosek.Task.getaslicetrip**

Syntax:

```
getaslicetrip (
    mosek.acemode accmode,
    int first,
    int last,
    List|None surp,
    array(int) subi,
    array(int) subj,
    array(double) val)
```

Arguments:

accmode (input) Defines whether a column-slice or a row-slice is requested.

first (input) Index of the first row or variable in the sequence.
last (input) Index of the last row or variable 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.

See also:

Task.getaslicenumnz Obtains the number of non-zeros in a row or column slice of A .

- **mosek.Task.getavec**

Syntax:

```
getavec (
    mosek.accmode accmode,
    int i,
    List|None nzi,
    array(int) subi,
    array(double) vali)
```

Arguments:

accmode (input) Defines if operations are performed row-wise (constraint-oriented) or column-wise (variable-oriented).
i (input) Index of the row or column.
nzi (output) Number of non-zeros in the vector obtained.
subi (output) Index of the non-zeros in the vector obtained.
vali (output) Numerical values of the vector to be 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:

```
int getavecnumnz (
    mosek.accmode accmode,
    int i)
```

Arguments:

- accmode (input)** Defines whether non-zeros are counted by columns or by rows.
- i (input)** Index of the row or column.

Returns:

Description: Obtains the number of non-zero elements in one row or column of A .

- `mosek.Task.getbound`

Syntax:

```
getbound (
    mosek.accmode accmode,
    int i,
    List|None bk,
    List|None bl,
    List|None bu)
```

Arguments:

- 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(int) bk,
    array(double) bl,
    array(double) bu)
```

Arguments:

- 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)
```

Arguments:

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:

```
double getcfix ()
```

Returns:

Description: Obtains the fixed term in the objective.

- `mosek.Task.getcone`

Syntax:

```
getcone (
    int k,
    List|None conetype,
    List|None coneapar,
    List|None nummem,
    array(int) submem)
```

Arguments:

k (input) Index of the cone 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.

submem (output) Variable subscripts of the members in the cone.

Description: Obtains a conic constraint.

- `mosek.Task.getconeinfo`

Syntax:

```
getconeinfo (
    int k,
    List|None conetype,
    List|None coneapar,
    List|None nummem)
```


Arguments:

- 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:

```
str getconname (int i)
```

Arguments:

- i (input)** Index.

Returns:

Description: Obtains a name of a constraint.

See also:

- Task.getmaxnamelen** Obtains the maximum length of any objective, constraint, variable or cone name.

- `mosek.Task.getcslice`

Syntax:

```
getcslice (
    int first,
    int last,
    array(double) c)
```

Arguments:

- 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.getdounf`

Syntax:

```
double getdounf (mosek.dinfitem whichdinf)
```

Arguments:

- whichdinf (input)** A double information item. See section 16.11 for the possible values.

Returns:

Description: Obtains a double information item from the task information database.

- `mosek.Task.getdouparam`

Syntax:

```
double getdouparam (mosek.dparam param)
```

Arguments:

`param` (**input**) Which parameter.

Returns:

Description: Obtains the value of a double parameter.

- `mosek.Task.getdualobj`

Syntax:

```
getdualobj (
    mosek.soltype whichsol,
    List|None dualobj)
```

Arguments:

`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:

```
Task getinfeasiblesubproblem (mosek.soltype whichsol)
```

Arguments:

`whichsol` (**input**) Which solution to use when determining the infeasible subproblem.

Returns:

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 9.

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.getintinf`

Syntax:

```
int getintinf (mosek.iinfitem whichiinf)
```

Arguments:

`whichiinf` (**input**) Specifies an information item.

Returns:

Description: Obtains an integer information item from the task information database.

- `mosek.Task.getintparam`

Syntax:

```
int getintparam (mosek.iparam param)
```

Arguments:

param (input) Which parameter.

Returns:

Description: Obtains the value of an integer parameter.

- `mosek.Task.getmaxnamelen`

Syntax:

```
getmaxnamelen (List|None maxlen)
```

Arguments:

maxlen (output) The maximum length of any name.

Description: Obtains the maximum length of any objective, constraint, variable or cone name.

- `mosek.Task.getmaxnumanz`

Syntax:

```
int getmaxnumanz ()
```

Returns:

Description: Obtains number of preallocated non-zeros for A . When this number of non-zeros is reached MOSEK will automatically allocate more space for A .

- `mosek.Task.getmaxnumcon`

Syntax:

```
getmaxnumcon (List|None maxnumcon)
```

Arguments:

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:

```
getmaxnumcone (List|None maxnumcone)
```

Arguments:

`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:

```
getmaxnumqnz (List|None maxnumqnz)
```

Arguments:

`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:

```
getmaxnumvar (List|None maxnumvar)
```

Arguments:

`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.getmemusagetask`

Syntax:

```
getmemusagetask (
    List|None meminuse,
    List|None maxmemuse)
```

Arguments:

`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:

```
str getname (
    mosek.problemitem whichitem,
    int i,
    List|None len)
```

Arguments:

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.

Returns:

Description: Obtains a name of a problem item, i.e. a cone, a variable or a constraint.

See also:

Task.getmaxnamelen Obtains the maximum length of any objective, constraint, variable or cone name.

- **mosek.Task.getnameindex**

Syntax:

```
int getnameindex (
    mosek.problemitem whichitem,
    str name,
    List|None asgn)
```

Arguments:

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.

Returns:

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.getnumanz**

Syntax:

```
int getnumanz ()
```

Returns:

Description: Obtains the number of non-zeros in *A*.

- **mosek.Task.getnumcon**

Syntax:

```
int getnumcon ()
```

Returns:

Description: Obtains the number of constraints.

- **mosek.Task.getnumcone**

Syntax:

```
int getnumcone ()
```

Returns:

Description: Obtains the number of cones.

- `mosek.Task.getnumconemem`

Syntax:

```
getnumconemem (
    int k,
    List|None nummem)
```

Arguments:

`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:

```
getnumintvar (List|None numintvar)
```

Arguments:

`numintvar` (**output**) Number of integer variables.

Description: Obtains the number of integer constrained variables.

- `mosek.Task.getnumparam`

Syntax:

```
getnumparam (
    mosek.parametertype partype,
    List|None numparam)
```

Arguments:

`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.getnumqconnz`

Syntax:

```
int getnumqconnz (int i)
```

Arguments:

`i` (**input**) Index of the constraint for which the quadratic terms should be obtained.

Returns:

Description: Obtains the number of non-zero quadratic terms in a constraint.

- `mosek.Task.getnumqobjnz`

Syntax:

```
int getnumqobjnz ()
```

Returns:

Description: Obtains the number of non-zero quadratic terms in the objective.

- `mosek.Task.getnumvar`

Syntax:

```
int getnumvar ()
```

Returns:

Description: Obtains the number of variables.

- `mosek.Task.getobjname`

Syntax:

```
str getobjname (List|None len)
```

Arguments:

`len (output)` Assigned the length of the objective name.

Returns:

Description: Obtains the name assigned to the objective function.

- `mosek.Task.getobjsense`

Syntax:

```
int getobjsense ()
```

Returns:

Description: Gets the objective sense of the task.

See also:

`Task.putobjsense` Sets the objective sense.

- `mosek.Task.getprimalobj`

Syntax:

```
double getprimalobj (mosek.soltype whichsol)
```

Arguments:

`whichsol (input)` Selects a solution.

Returns:

Description: Obtains the primal objective value for a solution.

- `mosek.Task.getproctype`

Syntax:

```
getproctype (List|None proctype)
```

Arguments:

proctype (output) The problem type.

Description: Obtains the problem type.

- `mosek.Task.getqconk`

Syntax:

```
int getqconk (
    int k,
    List|None qcsurp,
    array(int) qcsubi,
    array(int) qcsubj,
    array(double) qcval)
```

Arguments:

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 (5.36).

qcsubj (output) j subscripts for q_{ij}^k . See (5.36).

qcval (output) Numerical value for q_{ij}^k .

Returns:

Description: Obtains all the quadratic terms in a constraint. The quadratic terms are stored sequentially `qcsubi`, `qcsubj`, and `qcval`.

- `mosek.Task.getqobj`

Syntax:

```
getqobj (
    List|None qosurp,
    List|None numqonz,
    array(int) qosubi,
    array(int) qosubj,
    array(double) qoval)
```

Arguments:

qosurp (input/output) When entering the function `qosurp` `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.

numqonz (output) Number of non-zero elements in Q^o .

qosubi (output) i subscript for q_{ij}^o .

qosubj (output) j subscript for q_{ij}^o .

qoval (output) Numerical value for q_{ij}^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:

```
getqobjij (
    int i,
    int j,
    List|None qoij)
```

Arguments:

i (input) Row index of the coefficient.

j (input) Column index of coefficient.

qoij (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)
```

Arguments:

whichsol (input) Selects a solution.

first (input) See formula (13.5) for the definition.

last (input) See formula (13.5) 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.

$$\text{redcosts}[j - \text{first} + 0] = (s_l^x)_j - (s_u^x)_j, \quad j = \text{first}, \dots, \text{last} - 1. \quad (13.5)$$

- `mosek.Task.getsolution`

Syntax:

```

getsolution (
    mosek.soltype whichsol,
    List|None prosta,
    List|None solsta,
    array(int) skc,
    array(int) skx,
    array(int) 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)

```

Arguments:

whichsol (input) Selects a solution.

prosta (output) Problem status.

solsta (output) Solution status.

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 (s_u^x).

snx (output) Dual variables corresponding to the conic constraints on the variables (s_n^x).

Description: Obtains the complete solution.

Consider the case of linear programming. The primal problem is given by

$$\begin{aligned}
 &\text{minimize} && c^T x + c^f \\
 &\text{subject to} && \begin{array}{ccc} l^c & \leq & Ax & \leq & u^c, \\ l^x & \leq & x & \leq & u^x. \end{array}
 \end{aligned} \tag{13.6}$$

and the corresponding dual problem is

$$\begin{aligned}
 &\text{maximize} && (l^c)^T s_l^c - (u^c)^T s_u^c \\
 &&& + (l^x)^T s_l^x - (u^x)^T s_u^x + c^f \\
 &\text{subject to} && \begin{array}{rcl} A^T y + s_l^x - s_u^x & = & c, \\ -y + s_l^c - s_u^c & = & 0, \\ s_l^c, s_u^c, s_l^x, s_u^x & \geq & 0. \end{array}
 \end{aligned} \tag{13.7}$$

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_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:

```
getsolutioni (
    mosek.accmode accmode,
    int i,
    mosek.soltype whichsol,
    List|None sk,
    List|None x,
    List|None sl,
    List|None su,
    List|None sn)
```

Arguments:

accmode (input) Defines if operations are performed row-wise (constraint-oriented) or column-wise (variable-oriented).
i (input) Index of the constraint or variable.
whichsol (input) Selects a solution.
sk (output) Status key of the constraint of 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:

```
getsolutioninf (
    mosek.soltype whichsol,
    List|None prosta,
    List|None solsta,
    List|None primalobj,
    List|None maxpbi,
    List|None maxpcni,
    List|None maxpeqi,
    List|None maxinti,
    List|None dualobj,
    List|None maxdbi,
    List|None maxdcni,
    List|None maxdeqi)
```

Arguments:

whichsol (input) Selects a solution.

prosta (output) Problem status.

solsta (output) Solution status.

primalobj (output) Objective value corresponding to the primal solution.

maxpbi (output) Maximum primal bound infeasibility.

maxpcni (output) Maximum infeasibility in the primal conic constraints.

maxpeqi (output) Maximum infeasibility in the primal equality constraints.

maxinti (output) Maximum infeasibility in integer constraints.

dualobj (output) Objective value corresponding to the dual solution.

maxdbi (output) Maximum dual bound infeasibility.

maxdcni (output) Maximum infeasibility in the dual conic constraints.

maxdeqi (output) Maximum infeasibility in the dual equality constraints.

Description: Obtains information about a solution.

- `mosek.Task.getsolutionslice`

Syntax:

```

getsolutionslice (
    mosek.soltype whichsol,
    mosek.solitem solitem,
    int first,
    int last,
    array(double) values)

```

Arguments:

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.h. if $xx[5, \dots, 9]$ 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

$$\begin{aligned}
 & \text{minimize} && c^T x + c^f \\
 & \text{subject to} && l^c \leq Ax \leq u^c, \\
 & && l^x \leq x \leq u^x.
 \end{aligned} \tag{13.8}$$

and the corresponding dual problem is

$$\begin{aligned}
 & \text{maximize} && (l^c)^T s_l^c - (u^c)^T s_u^c \\
 & && + (l^x)^T s_l^x - (u^x)^T s_u^x + c^f \\
 & \text{subject to} && A^T y + s_l^x - s_u^x = c, \\
 & && -y + s_l^c - s_u^c = 0, \\
 & && s_l^c, s_u^c, s_l^x, s_u^x \geq 0.
 \end{aligned} \tag{13.9}$$

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:

$$\begin{aligned}
 & \text{maximize} && (l^c)^T s_l^c - (u^c)^T s_u^c \\
 & && + (l^x)^T s_l^x - (u^x)^T s_u^x + c^f \\
 & \text{subject to} && A^T y + s_l^x - s_u^x + s_n^x = c, \\
 & && -y + s_l^c - s_u^c = 0, \\
 & && s_l^c, s_u^c, s_l^x, s_u^x \geq 0, \\
 & && s_n^x \in \mathcal{C}^*
 \end{aligned} \tag{13.10}$$

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_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:

```
getsolutionstatus (
    mosek.soltype whichsol,
    List|None prosta,
    List|None solsta)
```

Arguments:

`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(int) sk)
```

Arguments:

`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,...,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.getstrparam**

Syntax:

```
str getstrparam (
    mosek.sparam param,
    List|None len)
```

Arguments:

param (input) Which parameter.

len (output) The length of the parameter value.

Returns:

Description: Obtains the value of a string parameter.

- **mosek.Task.gettaskname**

Syntax:

```
str gettaskname (List|None len)
```

Arguments:

len (output) Is assigned the length of the task name.

Returns:

Description: Obtains the name assigned to the task.

- **mosek.Task.getvarbranchdir**

Syntax:

```
int getvarbranchdir (int j)
```

Arguments:

j (input) Index of the variable.

Returns:

Description: Obtains the branching direction for a given variable *j*.

- **mosek.Task.getvarbranchpri**

Syntax:

```
int getvarbranchpri (int j)
```

Arguments:

j (input) Index of the variable.

Returns:

Description: Obtains the branching priority for a given variable *j*.

- `mosek.Task.getvarname`

Syntax:

```
str getvarname (int i)
```

Arguments:

i (input) Index.

Returns:

Description: Obtains a name of a variable.

See also:

`Task.getmaxnamelen` Obtains the maximum length of any objective, constraint, variable or cone name.

- `mosek.Task.getvartype`

Syntax:

```
int getvartype (int j)
```

Arguments:

j (input) Index of the variable.

Returns:

Description: Gets the variable type of one variable.

- `mosek.Task.getvartypelist`

Syntax:

```
getvartypelist (
    array(int) subj,
    array(int) vartype)
```

Arguments:

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)
```

Arguments:

basis (output) The array of basis indexes to use.

The array is interpreted as follows: If $\text{basis}[i] \leq \text{numcon} - 1$, 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.

- `mosek.Task.inputdata`

Syntax:

```
inputdata (
    int maxnumcon,
    int maxnumvar,
    array(double) c,
    double cfix,
    array(int) aptrb,
    array(int) aptre,
    array(int) asub,
    array(double) aval,
    array(int) bkc,
    array(double) blc,
    array(double) buc,
    array(int) bkc,
    array(double) blc,
    array(double) buc,
    array(int) bkc,
    array(double) blc,
    array(double) buc,
    array(int) bkc,
    array(double) blc,
    array(double) buc)
```

Arguments:

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 (5.37) and section 5.8.3.

aptre (input) Pointers to the last element + 1 in the rows or the columns of A . See (5.37) and section 5.8.3.

asub (input) Coefficient subscripts. See (5.37) and section 5.8.3.

aval (input) Coefficient values. See (5.37) 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.
bkx (**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:

```
isdoupname (
    str parname,
    List|None param)
```

Arguments:

parname (**input**) Parameter name.
param (**output**) Which parameter.

Description: Checks whether **parname** is a valid double parameter name.

- `mosek.Task.isintparname`

Syntax:

```
isintparname (
    str parname,
    List|None param)
```

Arguments:

parname (**input**) Parameter name.
param (**output**) Which parameter.

Description: Checks whether **parname** is a valid integer parameter name.

- `mosek.Task.isstrparname`

Syntax:

```
isstrparname (
    str parname,
    List|None param)
```

Arguments:

parname (**input**) Parameter name.
param (**output**) Which parameter.

Description: Checks whether **parname** is a valid string parameter name.

- `mosek.Task.linkfiletotaskstream`

Syntax:

```
linkfiletotaskstream (
    mosek.streamtype whichstream,
    str filename,
    int append)
```

Arguments:

`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)
```

Arguments:

`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.optimize`

Syntax:

```
optimize ()
```

Description: Calls the optimizer. Depending on the problem type and the selected solver this will call one of the solvers in MOSEK.

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.optimizeconcurrent`

Syntax:

```
optimizeconcurrent (array(Task) taskarray)
```

Arguments:

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 7.6.4.

- **mosek.Task.optimizetrm**

Syntax:

```
optimizetrm (List|None trmcode)
```

Arguments:

trmcode (output) Is either **mosek.rescode.ok** or a termination response code.

Description: This function is equivalent to **Task.optimize** except in the case where **Task.optimize** would have returned a termination response code such as

- **mosek.rescode.trm_max_iterations** or
- **mosek.rescode.trm_stall**.

- **mosek.Task.primalsensitivity**

Syntax:

```
primalsensitivity (
    array(int) subi,
    array(int) marki,
    array(int) subj,
    array(int) 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)

```

Arguments:

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 11.

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:

$$\begin{aligned}
&\text{minimize} && x_1 + x_2 \\
&\text{subject to} && -1 \leq x_1 - x_2 \leq 1, \\
&&& x_1 = 0, \\
&&& x_1 \geq 0, x_2 \geq 0
\end{aligned} \tag{13.11}$$

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:

$$x_1 - x_2 \leq 1 \tag{13.12}$$

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 11.5.

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)

```

Arguments:

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, then c is printed.

qo (input) If non-zero, then Q^o is printed.

a (input) If non-zero, then A is printed.

qc (input) If non-zero, then Q^k is printed for the relevant constraints.

bc (input) If non-zero, then constraints bounds are printed.

bx (input) If non-zero, then variable bounds are printed.

vartype (input) If non-zero, then variable types are printed.

cones (input) If non-zero, then 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)

```

Arguments:

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 A .

Task.putaijlist Changes one or more coefficients in A .

Comments:

- `mosek.Task.putaijlist`

Syntax:

```
putaijlist (
    array(int) subi,
    array(int) subj,
    array(double) valij)
```

Arguments:

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 A .

Task.putaij Changes a single value in the linear coefficient matrix.

Task.putmaxnumanz The function changes the size of the preallocated storage for linear coefficients.

Comments:

- `mosek.Task.putavec`

Syntax:

```
putavec (
    mosek.accmode accmode,
    int i,
    array(int) asub,
    array(double) aval)
```


Arguments:

- 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 5.8.3.2 and ??.

See also:

- Task.putaij** Changes a single value in the linear coefficient matrix.
- Task.putmaxnumanz** The function changes the size of the preallocated storage for linear coefficients.

- **mosek.Task.putaveclist**

Syntax:

```
putaveclist (
    mosek.accmode accmode,
    array(int) sub,
    array(int) ptrb,
    array(int) pre,
    array(int) asub,
    array(double) aval)
```

Arguments:

- 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 5.8.3.2 and section ??.

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 5.8.3.2 and section ??.

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 (5.37) 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 A .

Task.putaij Changes a single value in the linear coefficient matrix.

Task.putmaxnumanz The function changes the size of the preallocated storage for linear coefficients.

- **mosek.Task.putbound**

Syntax:

```
putbound (
    mosek.accmode accmode,
    int i,
    mosek.boundkey bk,
    double bl,
    double bu)
```

Arguments:

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.chgbound` Changes the bounds for one constraint or variable.

`Task.putboundlist` Changes the bounds of constraints or variables.

- `mosek.Task.putboundlist`

Syntax:

```
putboundlist (
    mosek.accmode accmode,
    array(int) sub,
    array(int) bk,
    array(double) bl,
    array(double) bu)
```

Arguments:

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) If `con` is non-zero (zero), then constraint (variable) `sub[t]` is assigned the key `bk[t]`.

bl (input) If `con` is non-zero (zero), then constraint (variable) `sub[t]` is assigned the lower bound `bl[t]`.

bu (input) If `con` is non-zero (zero), then constraint (variable) `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(int) bk,
    array(double) bl,
    array(double) bu)
```

Arguments:

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)
```

Arguments:

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)
```

Arguments:

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)
```

Arguments:

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)
```

Arguments:

`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)
```

Arguments:

`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)
```

Arguments:

`param (input)` Which parameter.

`parvalue (input)` Parameter value.

Description: Sets the value of an integer parameter.

- `mosek.Task.putmaxnumanz`

Syntax:

```
putmaxnumanz (int maxnumanz)
```

Arguments:

maxnumanz (input) New size of the storage reserved for storing *A*.

Description: MOSEK stores only the non-zero elements in *A*. Therefore, MOSEK cannot predict how much storage is required to store *A*. Using this function it is possible to specify the number of non-zeros to preallocate for storing *A*.

If the number of non-zeros in the problem is known, it is a good idea to set **maxnumanz** slightly larger than this number, otherwise a rough estimate can be used. In general, if *A* is inputted in many small chunks, setting this value may speed up the data input phase.

It is not mandatory to call this function, since MOSEK will reallocate internal structures whenever it is necessary.

See also:

```
mosek.iparam.maxnumanz_double_trh
mosek.iinfitem.sto_num_a_realloc
```

- `mosek.Task.putmaxnumcon`

Syntax:

```
putmaxnumcon (int maxnumcon)
```

Arguments:

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)
```

Arguments:

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 (int maxnumqnz)
```

Arguments:

`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 A 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 A .

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)
```

Arguments:

`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)
```

Arguments:

`paramname` (**input**) Name of a MOSEK parameter.

`parvalue` (**input**) Parameter value.

Description: Sets the value of a named double parameter.

- `mosek.Task.putnaintparam`

Syntax:

```

    putnaintparam (
        str paramname,
        int parvalue)

```

Arguments:

`paramname` (**input**) Name of a MOSEK 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)

```

Arguments:

`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)

```

Arguments:

`paramname` (**input**) Name of a MOSEK parameter.
`parvalue` (**input**) Parameter value.

Description: Sets the value of a named string parameter.

- `mosek.Task.putobjname`

Syntax:

```

    putobjname (str objname)

```

Arguments:

`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)
```

Arguments:

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)
```

Arguments:

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)
```

Arguments:

qcsbk (input) k subscripts for q_{ij}^k . See (5.36).

qcsubi (input) i subscripts for q_{ij}^k . See (5.36).

qcsubj (input) j subscripts for q_{ij}^k . See (5.36).

qcval (input) Numerical value for q_{ij}^k .

Description: Replaces all quadratic entries in the constraints. Consider constraints on the form:

$$l_k^c \leq \frac{1}{2} \sum_{i=0}^{\text{numvar}-1} \sum_{j=0}^{\text{numvar}-1} q_{ij}^k x_i x_j + \sum_{j=0}^{\text{numvar}-1} a_{kj} x_j \leq u_k^c, \quad k = 0, \dots, m-1. \quad (13.13)$$

The function assigns values to q such that:

$$q_{\text{qcsubi}[t], \text{qcsubj}[t]}^{\text{qcsubk}[t]} = \text{qcval}[t], \quad t = 0, \dots, \text{numqcnz} - 1. \quad (13.14)$$

and

$$q_{\text{qcsubj}[t], \text{qcsubi}[t]}^{\text{qcsubk}[t]} = \text{qcval}[t], \quad t = 0, \dots, \text{numqcnz} - 1. \quad (13.15)$$

Values not assigned are set to zero.

See also:

Task.putqcnk Replaces all quadratic terms in a single constraint.

Task.putmaxnumqnz Changes the size of the preallocated storage for Q .

- **mosek.Task.putqcnk**

Syntax:

```
putqcnk (
    int k,
    array(int) qcsubi,
    array(int) qcsubj,
    array(double) qcval)
```

Arguments:

k (input) The constraint in which new the Q elements are inserted.

qcsubi (input) i subscripts for q_{ij}^k . See (5.36).

qcsubj (input) j subscripts for q_{ij}^k . See (5.36).

qcval (input) Numerical value for q_{ij}^k .

Description: Replaces all the quadratic entries in one constraint k of the form:

$$l_k^c \leq \frac{1}{2} \sum_{i=0}^{\text{numvar}-1} \sum_{j=0}^{\text{numvar}-1} q_{ij}^k x_i x_j + \sum_{j=0}^{\text{numvar}-1} a_{kj} x_j \leq u_k^c. \quad (13.16)$$

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 $\text{numqcnz} - 1$
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.putmaxnumqnz** 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.

Task.putmaxnumqnz Changes the size of the preallocated storage for Q .

- **mosek.Task.putqobj**

Syntax:

```
putqobj (
    array(int) qosubi,
    array(int) qosubj,
    array(double) qoval)
```

Arguments:

`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

$$\frac{1}{2} \sum_{i=0}^{\text{numvar}-1} \sum_{j=0}^{\text{numvar}-1} q_{ij}^o x_i x_j + \sum_{j=0}^{\text{numvar}-1} c_j x_j + c^f. \quad (13.17)$$

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 `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 elements are *not* added to together; only the last value for each entry is used.

For a code example see section 5.3.1.

- **mosek.Task.putqobjij**

Syntax:

```
putqobjij (
    int i,
    int j,
    double qoij)
```

Arguments:

- 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 of the coefficients in the quadratic term in the objective. The function performs the assignment

$$q_{ij}^o = \text{qoij}.$$

- **mosek.Task.putsolution**

Syntax:

```
putsolution (
    mosek.soltype whichsol,
    array(int) skc,
    array(int) skx,
    array(int) 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)
```

Arguments:

- 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 (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)
```

Arguments:

- accmode (input)** If non-zero, then 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 function 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)
```

Arguments:

- 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)
```

Arguments:

`param` (**input**) Which parameter.

`parvalue` (**input**) Parameter value.

Description: Sets the value of a string parameter.

- `mosek.Task.puttaskname`

Syntax:

```
puttaskname (str taskname)
```

Arguments:

`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)
```

Arguments:

`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)

```

Arguments:

j (input) Index of the variable.
vartype (input) The new variable type.

Description: Sets the variable type of one variable.

- `mosek.Task.putvartypelist`

Syntax:

```

    putvartypelist (
        array(int) subj,
        array(int) vartype)

```

Arguments:

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 16.48 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.

- `mosek.Task.readbranchpriorities`

Syntax:

```

    readbranchpriorities (str filename)

```

Arguments:

filename (input) Data is written to 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)

```

Arguments:

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” is interpreted as an OPF file and a compressed OPF file respectively.
- “.mps” and “.mps.gz” is interpreted as an MPS file and a compressed MPS file respectively.
- “.mbt” is interpreted as an MBT file (MOSEK binary task file).

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)
```

Arguments:

`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)
```

Arguments:

`whichstream` (**input**) Index of the stream.

Description: Prints a short summary of last file that was read.

- `mosek.Task.relaxprimal`

Syntax:


```

Task relaxprimal (
    array(double) wlc,
    array(double) wuc,
    array(double) wlx,
    array(double) wux)

```

Arguments:

- 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.

Returns:

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

$$\begin{aligned}
 & \text{minimize} && c^T x \\
 & \text{subject to} && l^c \leq Ax \leq u^c, \\
 & && l^x \leq x \leq u^x,
 \end{aligned} \tag{13.18}$$

the function forms a new task **relaxedtask** having the form

$$\begin{aligned}
 & \text{minimize} && p \\
 & \text{subject to} && l^c \leq Ax + v_l^c - v_u^c \leq u^c, \\
 & && l^x \leq x + v_l^x - v_u^x \leq u^x, \\
 & && (w_l^c)^T v_l^c + (w_u^c)^T v_u^c + (w_l^x)^T v_l^x + (w_u^x)^T v_u^x - p \leq 0, \\
 & && v_l^c, v_u^c, v_l^x, v_u^x \geq 0.
 \end{aligned} \tag{13.19}$$

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 (13.19) 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 (13.19) 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 (13.19), then a natural thing to do is to solve the optimization problem

$$\begin{aligned}
 & \text{minimize} && c^T x \\
 & \text{subject to} && l^c \leq Ax + v_l^c - v_u^c \leq u^c, \\
 & && l^x \leq x + v_l^x - v_u^x \leq u^x, \\
 & && (w_l^c)^T v_l^c + (w_u^c)^T v_u^c + (w_l^x)^T v_l^x + (w_u^x)^T v_u^x - p \leq 0, \\
 & && p = p^*, \\
 & && v_l^c, v_u^c, v_l^x, v_u^x \geq 0,
 \end{aligned} \tag{13.20}$$

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 (13.19) or the problem (13.20). The parameter can take one of the following values.

`mosek.feasrepairtype.optimize_none` : The returned task `relaxedtask` contains problem (13.19) and is not optimized.

`mosek.feasrepairtype.optimize_penalty` : The returned task `relaxedtask` contains problem (13.19) and is optimized.

`mosek.feasrepairtype.optimize_combined` : The returned task `relaxedtask` contains problem (13.20) and is optimized.

Please note that the v variables are appended to the x variables ordered as

$$(v_u^c)_1, (v_l^c)_1, (v_u^c)_2, (v_l^c)_2, \dots, (v_u^c)_m, (v_l^c)_m, \quad (v_u^x)_1, (v_l^x)_1, (v_u^x)_2, (v_l^x)_2, \dots, (v_u^x)_n, (v_l^x)_n$$

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)
```

Arguments:

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)
```

Arguments:

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)
```

Arguments:

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.

`Task.putmaxnumanz` The function changes the size of the preallocated storage for linear coefficients.

`Task.putmaxnumqnz` Changes the size of the preallocated storage for Q .

- `mosek.Task.sensitivityreport`

Syntax:

```
sensitivityreport (mosek.streamtype whichstream)
```

Arguments:

`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)
```

Arguments:

`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:

```
int solutiondef (mosek.soltype whichsol)
```

Arguments:

whichsol (input) Selects a solution.

Returns:

Description: Checks whether a solution is defined.

- `mosek.Task.solutionsummary`

Syntax:

```
solutionsummary (mosek.streamtype whichstream)
```

Arguments:

whichstream (input) Index of the stream.

Description: Prints a short summary of the current solution.

- `mosek.Task.solvewithbasis`

Syntax:

```
int solvewithbasis (
    int transp,
    int numnz,
    array(int) sub,
    array(double) val)
```

Arguments:

transp (input) If this argument is non-zero, then (13.22) is solved. Otherwise the system (13.21) 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.

Returns:

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

$$B\bar{x} = b \quad (13.21)$$

or the system

$$B^T\bar{x} = b \quad (13.22)$$

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 use with the **Task.solvewithbasis** function.

- **mosek.Task.startstat**

Syntax:

`startstat ()`

Description: Starts the statistics file.

- **mosek.Task.stopstat**

Syntax:

`stopstat ()`

Description: Stops the statistics file.

- `mosek.Task.strtoconetype`

Syntax:

```
strtoconetype (
    str str,
    List|None conetype)
```

Arguments:

`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:

```
strtosk (
    str str,
    List|None sk)
```

Arguments:

`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)
```

Arguments:

`whichsol` (**input**) Selects a solution.

Description: Undefines a solution. Purges all information regarding `whichsol`.

- `mosek.Task.writebranchpriorities`

Syntax:

```
writebranchpriorities (str filename)
```

Arguments:

`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)
```

Arguments:

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 B.

MPS : A text based column oriented format. File extension `.mps`. See Appendix A.

OPF : A text based row oriented format. File extension `.opf`. Supports more problem types than MPS and LP. See Appendix C.

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.

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)
```

Arguments:

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)
```

Arguments:

whichsol (input) Selects a solution.

filename (input) A valid file name.

Description: Saves the current basic, interior-point, or integer solution to a file.

13.7 Class mosek.Warning

Derived from:

`mosek.Exception`

Description:

This is an exception class representing MOSEK warnings.

13.7.1 Constructors

- `mosek.Warning`

Syntax:

```
Warning (mosek.rescode code)
```

Description:

Construct a warning from a MOSEK error code.

Arguments:

code The MOSEK response code to create the exception from.

Chapter 14

Parameter reference

14.1 Parameter groups

Parameters grouped by meaning and functionality.

14.1.1 Logging parameters.

- `mosek.iparam.log` 289
Controls the amount of log information.
- `mosek.iparam.log_bi` 289
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` 290
Controls the logging frequency.
- `mosek.iparam.log_concurrent` 290
Controls amount of output printed by the concurrent optimizer.
- `mosek.iparam.log_cut_second_opt` 290
Controls the reduction in the log levels for the second and any subsequent optimizations.
- `mosek.iparam.log_factor` 291
If turned on, then the factor log lines are added to the log.
- `mosek.iparam.log_feasrepair` 291
Controls the amount of output printed when performing feasibility repair.
- `mosek.iparam.log_file` 291
If turned on, then some log info is printed when a file is written or read.

• <code>mosek.iparam.log_head</code>	291
If turned on, then a header line is added to the log.	
• <code>mosek.iparam.log_infeas_ana</code>	292
Controls log level for the infeasibility analyzer.	
• <code>mosek.iparam.log_intpnt</code>	292
Controls the amount of log information from the interior-point optimizers.	
• <code>mosek.iparam.log_mio</code>	292
Controls the amount of log information from the mixed-integer optimizers.	
• <code>mosek.iparam.log_mio_freq</code>	292
The mixed integer solver logging frequency.	
• <code>mosek.iparam.log_nonconvex</code>	293
Controls amount of output printed by the nonconvex optimizer.	
• <code>mosek.iparam.log_optimizer</code>	293
Controls the amount of general optimizer information that is logged.	
• <code>mosek.iparam.log_order</code>	293
If turned on, then factor lines are added to the log.	
• <code>mosek.iparam.log_param</code>	293
Controls the amount of information printed out about parameter changes.	
• <code>mosek.iparam.log_presolve</code>	294
Controls amount of output printed by the presolve procedure. A higher level implies that more information is logged.	
• <code>mosek.iparam.log_response</code>	294
Controls amount of output printed when response codes are reported. A higher level implies that more information is logged.	
• <code>mosek.iparam.log_sensitivity</code>	294
Control logging in sensitivity analyzer.	
• <code>mosek.iparam.log_sensitivity_opt</code>	295
Control logging in sensitivity analyzer.	
• <code>mosek.iparam.log_sim</code>	295
Controls the amount of log information from the simplex optimizers.	
• <code>mosek.iparam.log_sim_freq</code>	295
Controls simplex logging frequency.	
• <code>mosek.iparam.log_sim_network_freq</code>	296
Controls the network simplex logging frequency.	
• <code>mosek.iparam.log_storage</code>	296
Controls the memory related log information.	

14.1.2 Basis identification parameters.

- `mosek.iparam.bi_clean_optimizer` 276
Controls which simplex optimizer is used in the clean-up phase.
- `mosek.iparam.bi_ignore_max_iter` 276
Turns on basis identification in case the interior-point optimizer is terminated due to maximum number of iterations.
- `mosek.iparam.bi_ignore_num_error` 277
Turns on basis identification in case the interior-point optimizer is terminated due to a numerical problem.
- `mosek.dparam.bi_lu_tol_rel_piv` 249
Relative pivot tolerance used in the LU factorization in the basis identification procedure.
- `mosek.iparam.bi_max_iterations` 277
Maximum number of iterations after basis identification.
- `mosek.iparam.intpnt_basis` 283
Controls whether basis identification is performed.
- `mosek.iparam.log_bi` 289
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` 290
Controls the logging frequency.

14.1.3 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` 276
Turns on basis identification in case the interior-point optimizer is terminated due to maximum number of iterations.
- `mosek.iparam.bi_ignore_num_error` 277
Turns on basis identification in case the interior-point optimizer is terminated due to a numerical problem.
- `mosek.iparam.intpnt_basis` 283
Controls whether basis identification is performed.
- `mosek.dparam.intpnt_co_tol_dfeas` 252
Dual feasibility tolerance used by the conic interior-point optimizer.
- `mosek.dparam.intpnt_co_tol_infeas` 252
Infeasibility tolerance for the conic solver.

• <code>mosek.dparam.intpnt_co_tol_mu_red</code>	252
Optimality tolerance for the conic solver.	
• <code>mosek.dparam.intpnt_co_tol_near_rel</code>	253
Optimality tolerance for the conic solver.	
• <code>mosek.dparam.intpnt_co_tol_pfeas</code>	253
Primal feasibility tolerance used by the conic interior-point optimizer.	
• <code>mosek.dparam.intpnt_co_tol_rel_gap</code>	253
Relative gap termination tolerance used by the conic interior-point optimizer.	
• <code>mosek.iparam.intpnt_diff_step</code>	283
Controls whether different step sizes are allowed in the primal and dual space.	
• <code>mosek.iparam.intpnt_max_iterations</code>	284
Controls the maximum number of iterations allowed in the interior-point optimizer.	
• <code>mosek.iparam.intpnt_max_num_cor</code>	284
Maximum number of correction steps.	
• <code>mosek.iparam.intpnt_max_num_refinement_steps</code>	284
Maximum number of steps to be used by the iterative search direction refinement.	
• <code>mosek.dparam.intpnt_nl_merit_bal</code>	253
Controls if the complementarity and infeasibility is converging to zero at about equal rates.	
• <code>mosek.dparam.intpnt_nl_tol_dfeas</code>	254
Dual feasibility tolerance used when a nonlinear model is solved.	
• <code>mosek.dparam.intpnt_nl_tol_mu_red</code>	254
Relative complementarity gap tolerance.	
• <code>mosek.dparam.intpnt_nl_tol_near_rel</code>	254
Nonlinear solver optimality tolerance parameter.	
• <code>mosek.dparam.intpnt_nl_tol_pfeas</code>	255
Primal feasibility tolerance used when a nonlinear model is solved.	
• <code>mosek.dparam.intpnt_nl_tol_rel_gap</code>	255
Relative gap termination tolerance for nonlinear problems.	
• <code>mosek.dparam.intpnt_nl_tol_rel_step</code>	255
Relative step size to the boundary for general nonlinear optimization problems.	
• <code>mosek.iparam.intpnt_off_col_trh</code>	285
Controls the aggressiveness of the offending column detection.	
• <code>mosek.iparam.intpnt_order_method</code>	285
Controls the ordering strategy.	
• <code>mosek.iparam.intpnt_regularization_use</code>	286
Controls whether regularization is allowed.	

- `mosek.iparam.intpnt_scaling` 286
Controls how the problem is scaled before the interior-point optimizer is used.
- `mosek.iparam.intpnt_solve_form` 286
Controls whether the primal or the dual problem is solved.
- `mosek.iparam.intpnt_starting_point` 287
Starting point used by the interior-point optimizer.
- `mosek.dparam.intpnt_tol_dfeas` 255
Dual feasibility tolerance used for linear and quadratic optimization problems.
- `mosek.dparam.intpnt_tol_dsafe` 255
Controls the interior-point dual starting point.
- `mosek.dparam.intpnt_tol_infeas` 256
Nonlinear solver infeasibility tolerance parameter.
- `mosek.dparam.intpnt_tol_mu_red` 256
Relative complementarity gap tolerance.
- `mosek.dparam.intpnt_tol_path` 256
interior-point centering aggressiveness.
- `mosek.dparam.intpnt_tol_pfeas` 257
Primal feasibility tolerance used for linear and quadratic optimization problems.
- `mosek.dparam.intpnt_tol_psafe` 257
Controls the interior-point primal starting point.
- `mosek.dparam.intpnt_tol_rel_gap` 257
Relative gap termination tolerance.
- `mosek.dparam.intpnt_tol_rel_step` 257
Relative step size to the boundary for linear and quadratic optimization problems.
- `mosek.dparam.intpnt_tol_step_size` 258
If the step size falls below the value of this parameter, then the interior-point optimizer assumes it is stalled. If it does not make any progress.
- `mosek.iparam.log_concurrent` 290
Controls amount of output printed by the concurrent optimizer.
- `mosek.iparam.log_intpnt` 292
Controls the amount of log information from the interior-point optimizers.
- `mosek.iparam.log_presolve` 294
Controls amount of output printed by the presolve procedure. A higher level implies that more information is logged.

14.1.4 Simplex optimizer parameters.

Parameters defining the behavior of the simplex optimizer for linear problems.

• <code>mosek.dparam.basis_rel_tol_s</code>	248
Maximum relative dual bound violation allowed in an optimal basic solution.	
• <code>mosek.dparam.basis_tol_s</code>	249
Maximum absolute dual bound violation in an optimal basic solution.	
• <code>mosek.dparam.basis_tol_x</code>	249
Maximum absolute primal bound violation allowed in an optimal basic solution.	
• <code>mosek.iparam.log_sim</code>	295
Controls the amount of log information from the simplex optimizers.	
• <code>mosek.iparam.log_sim_freq</code>	295
Controls simplex logging frequency.	
• <code>mosek.iparam.log_sim_minor</code>	295
Currently not in use.	
• <code>mosek.iparam.sensitivity_optimizer</code>	317
Controls which optimizer is used for optimal partition sensitivity analysis.	
• <code>mosek.iparam.sim_degen</code>	318
Controls how aggressive degeneration is approached.	
• <code>mosek.iparam.sim_hotstart</code>	319
Controls the type of hot-start that the simplex optimizer perform.	
• <code>mosek.iparam.sim_max_iterations</code>	319
Maximum number of iterations that can be used by a simplex optimizer.	
• <code>mosek.iparam.sim_max_num_setbacks</code>	320
Controls how many setbacks that are allowed within a simplex optimizer.	
• <code>mosek.iparam.sim_network_detect_method</code>	321
Controls which type of detection method the network extraction should use.	
• <code>mosek.iparam.sim_non_singular</code>	321
Controls if the simplex optimizer ensures a non-singular basis, if possible.	
• <code>mosek.iparam.sim_save_lu</code>	323
Controls if the LU factorization stored should be replaced with the LU factorization corresponding to the initial basis.	
• <code>mosek.iparam.sim_scaling</code>	323
Controls how the problem is scaled before a simplex optimizer is used.	
• <code>mosek.iparam.sim_solve_form</code>	323
Controls whether the primal or the dual problem is solved by the primal-/dual- simplex optimizer.	

- `mosek.iparam.sim_stability_priority` 324
Controls how high priority the numerical stability should be given.
- `mosek.iparam.sim_switch_optimizer` 324
Controls the simplex behavior.
- `mosek.dparam.simplex_abs_tol_piv` 264
Absolute pivot tolerance employed by the simplex optimizers.

14.1.5 Primal simplex optimizer parameters.

Parameters defining the behavior of the primal simplex optimizer for linear problems.

- `mosek.iparam.sim_primal_crash` 321
Controls the simplex crash.
- `mosek.iparam.sim_primal_restrict_selection` 322
Controls how aggressively restricted selection is used.
- `mosek.iparam.sim_primal_selection` 322
Controls the primal simplex strategy.

14.1.6 Dual simplex optimizer parameters.

Parameters defining the behavior of the dual simplex optimizer for linear problems.

- `mosek.iparam.sim_dual_crash` 318
Controls whether crashing is performed in the dual simplex optimizer.
- `mosek.iparam.sim_dual_restrict_selection` 318
Controls how aggressively restricted selection is used.
- `mosek.iparam.sim_dual_selection` 319
Controls the dual simplex strategy.

14.1.7 Network simplex optimizer parameters.

Parameters defining the behavior of the network simplex optimizer for linear problems.

- `mosek.iparam.log_sim_network_freq` 296
Controls the network simplex logging frequency.
- `mosek.iparam.sim_network_detect` 320
Level of aggressiveness of network detection.
- `mosek.iparam.sim_network_detect_hotstart` 320
Level of aggressiveness of network detection in a simplex hot-start.

- `mosek.iparam.sim_refactor_freq` 322
Controls the basis refactoring frequency.

14.1.8 Nonlinear convex method parameters.

Parameters defining the behavior of the interior-point method for nonlinear convex problems.

- `mosek.iparam.check_convexity` 278
Specify the level of convexity check on quadratic problems
- `mosek.dparam.intpnt_nl_merit_bal` 253
Controls if the complementarity and infeasibility is converging to zero at about equal rates.
- `mosek.dparam.intpnt_nl_tol_dfeas` 254
Dual feasibility tolerance used when a nonlinear model is solved.
- `mosek.dparam.intpnt_nl_tol_mu_red` 254
Relative complementarity gap tolerance.
- `mosek.dparam.intpnt_nl_tol_near_rel` 254
Nonlinear solver optimality tolerance parameter.
- `mosek.dparam.intpnt_nl_tol_pfeas` 255
Primal feasibility tolerance used when a nonlinear model is solved.
- `mosek.dparam.intpnt_nl_tol_rel_gap` 255
Relative gap termination tolerance for nonlinear problems.
- `mosek.dparam.intpnt_nl_tol_rel_step` 255
Relative step size to the boundary for general nonlinear optimization problems.
- `mosek.dparam.intpnt_tol_infeas` 256
Nonlinear solver infeasibility tolerance parameter.

14.1.9 The conic interior-point method parameters.

Parameters defining the behavior of the interior-point method for conic problems.

- `mosek.dparam.intpnt_co_tol_dfeas` 252
Dual feasibility tolerance used by the conic interior-point optimizer.
- `mosek.dparam.intpnt_co_tol_infeas` 252
Infeasibility tolerance for the conic solver.
- `mosek.dparam.intpnt_co_tol_mu_red` 252
Optimality tolerance for the conic solver.
- `mosek.dparam.intpnt_co_tol_near_rel` 253
Optimality tolerance for the conic solver.

- `mosek.dparam.intpnt_co_tol_pfeas` 253
Primal feasibility tolerance used by the conic interior-point optimizer.
- `mosek.dparam.intpnt_co_tol_rel_gap` 253
Relative gap termination tolerance used by the conic interior-point optimizer.

14.1.10 The mixed integer optimization parameters.

- `mosek.iparam.log_mio` 292
Controls the amount of log information from the mixed-integer optimizers.
- `mosek.iparam.log_mio_freq` 292
The mixed integer solver logging frequency.
- `mosek.iparam.mio_branch_dir` 297
Controls whether the mixed integer optimizer is branching up or down by default.
- `mosek.iparam.mio_branch_priorities_use` 297
Controls whether branching priorities are used by the mixed integer optimizer.
- `mosek.iparam.mio_construct_sol` 297
Controls if initial MIP solution should be constructed from value of integer variables.
- `mosek.iparam.mio_cont_sol` 298
Controls the meaning of interior-point and basic solutions in MIP problems.
- `mosek.iparam.mio_cut_level_root` 298
Controls the cut level employed by the mixed integer optimizer at the root node.
- `mosek.iparam.mio_cut_level_tree` 299
Controls the cut level employed by the mixed integer optimizer in the tree.
- `mosek.dparam.mio_disable_term_time` 258
Certain termination criterias is disabled within the mixed integer optimizer for period time specified by the parameter.
- `mosek.iparam.mio_feaspump_level` 299
Controls the feasibility pump heuristic which is used to construct a good initial feasible solution.
- `mosek.iparam.mio_heuristic_level` 299
Controls the heuristic employed by the mixed integer optimizer to locate an initial integer feasible solution.
- `mosek.dparam.mio_heuristic_time` 259
Time limit for the mixed integer heuristics.
- `mosek.iparam.mio_keep_basis` 300
Controls whether the integer presolve keeps bases in memory.
- `mosek.iparam.mio_max_num_branches` 300
Maximum number of branches allowed during the branch and bound search.

• <code>mosek.iparam.mio_max_num_relaxs</code>	301
Maximum number of relaxations in branch and bound search.	
• <code>mosek.iparam.mio_max_num_solutions</code>	301
Controls how many feasible solutions the mixed-integer optimizer investigates.	
• <code>mosek.dparam.mio_max_time</code>	259
Time limit for the mixed integer optimizer.	
• <code>mosek.dparam.mio_max_time_aprx_opt</code>	259
Time limit for the mixed integer optimizer.	
• <code>mosek.dparam.mio_near_tol_abs_gap</code>	260
Relaxed absolute optimality tolerance employed by the mixed integer optimizer.	
• <code>mosek.dparam.mio_near_tol_rel_gap</code>	260
The mixed integer optimizer is terminated when this tolerance is satisfied.	
• <code>mosek.iparam.mio_node_optimizer</code>	302
Controls which optimizer is employed at the non-root nodes in the mixed integer optimizer.	
• <code>mosek.iparam.mio_node_selection</code>	302
Controls the node selection strategy employed by the mixed integer optimizer.	
• <code>mosek.iparam.mio_presolve_aggregate</code>	303
Controls whether problem aggregation is performed in the mixed integer presolve.	
• <code>mosek.iparam.mio_presolve_probing</code>	303
Controls whether probing is employed by the mixed integer presolve.	
• <code>mosek.iparam.mio_presolve_use</code>	303
Controls whether presolve is performed by the mixed integer optimizer.	
• <code>mosek.dparam.mio_rel_add_cut_limited</code>	260
Controls cut generation for MIP solver.	
• <code>mosek.iparam.mio_root_optimizer</code>	303
Controls which optimizer is employed at the root node in the mixed integer optimizer.	
• <code>mosek.iparam.mio_strong_branch</code>	304
The depth from the root in which strong branching is employed.	
• <code>mosek.dparam.mio_tol_abs_gap</code>	261
Absolute optimality tolerance employed by the mixed integer optimizer.	
• <code>mosek.dparam.mio_tol_abs_relax_int</code>	261
Integer constraint tolerance.	
• <code>mosek.dparam.mio_tol_rel_gap</code>	261
Relative optimality tolerance employed by the mixed integer optimizer.	
• <code>mosek.dparam.mio_tol_rel_relax_int</code>	261
Integer constraint tolerance.	

- `mosek.dparam.mio_tol_x` 262
Absolute solution tolerance used in mixed-integer optimizer.

14.1.11 Presolve parameters.

- `mosek.iparam.presolve_elim_fill` 308
Maximum amount of fill-in in the elimination phase.
- `mosek.iparam.presolve_eliminator_use` 309
Controls whether free or implied free variables are eliminated from the problem.
- `mosek.iparam.presolve_level` 309
Currently not used.
- `mosek.iparam.presolve_lindep_use` 309
Controls whether the linear constraints are checked for linear dependencies.
- `mosek.iparam.presolve_lindep_work_lim` 309
Controls linear dependency check in presolve.
- `mosek.dparam.presolve_tol_aij` 263
Absolute zero tolerance employed for constraint coefficients in the presolve.
- `mosek.dparam.presolve_tol_lin_dep` 263
Controls when a constraint is determined to be linearly dependent.
- `mosek.dparam.presolve_tol_s` 263
Absolute zero tolerance employed for slack variables in the presolve.
- `mosek.dparam.presolve_tol_x` 263
Absolute zero tolerance employed for variables in the presolve.
- `mosek.iparam.presolve_use` 310
Controls whether the presolve is applied to a problem before it is optimized.

14.1.12 Termination criterion parameters.

Parameters which define termination and optimality criteria and related information.

- `mosek.dparam.basis_rel_tol_s` 248
Maximum relative dual bound violation allowed in an optimal basic solution.
- `mosek.dparam.basis_tol_s` 249
Maximum absolute dual bound violation in an optimal basic solution.
- `mosek.dparam.basis_tol_x` 249
Maximum absolute primal bound violation allowed in an optimal basic solution.

• <code>mosek.iparam.bi_max_iterations</code>	277
Maximum number of iterations after basis identification.	
• <code>mosek.dparam.intpnt_co_tol_dfeas</code>	252
Dual feasibility tolerance used by the conic interior-point optimizer.	
• <code>mosek.dparam.intpnt_co_tol_infeas</code>	252
Infeasibility tolerance for the conic solver.	
• <code>mosek.dparam.intpnt_co_tol_mu_red</code>	252
Optimality tolerance for the conic solver.	
• <code>mosek.dparam.intpnt_co_tol_near_rel</code>	253
Optimality tolerance for the conic solver.	
• <code>mosek.dparam.intpnt_co_tol_pfeas</code>	253
Primal feasibility tolerance used by the conic interior-point optimizer.	
• <code>mosek.dparam.intpnt_co_tol_rel_gap</code>	253
Relative gap termination tolerance used by the conic interior-point optimizer.	
• <code>mosek.iparam.intpnt_max_iterations</code>	284
Controls the maximum number of iterations allowed in the interior-point optimizer.	
• <code>mosek.dparam.intpnt_nl_tol_dfeas</code>	254
Dual feasibility tolerance used when a nonlinear model is solved.	
• <code>mosek.dparam.intpnt_nl_tol_mu_red</code>	254
Relative complementarity gap tolerance.	
• <code>mosek.dparam.intpnt_nl_tol_near_rel</code>	254
Nonlinear solver optimality tolerance parameter.	
• <code>mosek.dparam.intpnt_nl_tol_pfeas</code>	255
Primal feasibility tolerance used when a nonlinear model is solved.	
• <code>mosek.dparam.intpnt_nl_tol_rel_gap</code>	255
Relative gap termination tolerance for nonlinear problems.	
• <code>mosek.dparam.intpnt_tol_dfeas</code>	255
Dual feasibility tolerance used for linear and quadratic optimization problems.	
• <code>mosek.dparam.intpnt_tol_infeas</code>	256
Nonlinear solver infeasibility tolerance parameter.	
• <code>mosek.dparam.intpnt_tol_mu_red</code>	256
Relative complementarity gap tolerance.	
• <code>mosek.dparam.intpnt_tol_pfeas</code>	257
Primal feasibility tolerance used for linear and quadratic optimization problems.	
• <code>mosek.dparam.intpnt_tol_rel_gap</code>	257
Relative gap termination tolerance.	

- `mosek.dparam.lower_obj_cut` 258
Objective bound.
- `mosek.dparam.lower_obj_cut_finite_trh` 258
Objective bound.
- `mosek.dparam.mio_disable_term_time` 258
Certain termination criterias is disabled within the mixed integer optimizer for period time specified by the parameter.
- `mosek.iparam.mio_max_num_branches` 300
Maximum number of branches allowed during the branch and bound search.
- `mosek.iparam.mio_max_num_solutions` 301
Controls how many feasible solutions the mixed-integer optimizer investigates.
- `mosek.dparam.mio_max_time` 259
Time limit for the mixed integer optimizer.
- `mosek.dparam.mio_near_tol_rel_gap` 260
The mixed integer optimizer is terminated when this tolerance is satisfied.
- `mosek.dparam.mio_tol_rel_gap` 261
Relative optimality tolerance employed by the mixed integer optimizer.
- `mosek.dparam.optimizer_max_time` 262
Solver time limit.
- `mosek.iparam.sim_max_iterations` 319
Maximum number of iterations that can be used by a simplex optimizer.
- `mosek.dparam.upper_obj_cut` 264
Objective bound.
- `mosek.dparam.upper_obj_cut_finite_trh` 264
Objective bound.

14.1.13 Progress call-back parameters.

- `mosek.dparam.callback_freq` 249
Controls progress call-back frequency.
- `mosek.iparam.solution_callback` 326
Indicates whether solution call-backs will be performed during the optimization.

14.1.14 Non-convex solver parameters.

- `mosek.iparam.log_nonconvex` 293
Controls amount of output printed by the nonconvex optimizer.
- `mosek.iparam.nonconvex_max_iterations` 304
Maximum number of iterations that can be used by the nonconvex optimizer.
- `mosek.dparam.nonconvex_tol_feas` 262
Feasibility tolerance used by the nonconvex optimizer.
- `mosek.dparam.nonconvex_tol_opt` 262
Optimality tolerance used by the nonconvex optimizer.

14.1.15 Feasibility repair parameters.

- `mosek.dparam.feasrepair_tol` 252
Tolerance for constraint enforcing upper bound on sum of weighted violations in feasibility repair.

14.1.16 Optimization system parameters.

Parameters defining the overall solver system environment. This includes system and platform related information and behavior.

- `mosek.iparam.cache_size_l1` 277
Specifies the size of the level 1 cache of the processor.
- `mosek.iparam.cache_size_l2` 278
Specifies the size of the level 2 cache of the processor.
- `mosek.iparam.check_ctrl_c` 278
Turns ctrl-c check on or off.
- `mosek.iparam.cpu_type` 280
Specifies the CPU type.
- `mosek.iparam.intpnt_num_threads` 285
Controls the number of threads employed by the interior-point optimizer.
- `mosek.iparam.license_cache_time` 287
Controls the license manager client behavior.
- `mosek.iparam.license_check_time` 288
Controls the license manager client behavior.
- `mosek.iparam.license_wait` 289
Controls if MOSEK should queue for a license if none is available.
- `mosek.iparam.log_storage` 296
Controls the memory related log information.

14.1.17 Output information parameters.

- `mosek.iparam.flush_stream_freq` 281
Controls the stream flushing frequency.
- `mosek.iparam.infeas_report_level` 282
Controls the contents of the infeasibility report.
- `mosek.iparam.license_suppress_expire_wrns` 288
Controls license manager client behavior.
- `mosek.iparam.log` 289
Controls the amount of log information.
- `mosek.iparam.log_bi` 289
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` 290
Controls the logging frequency.
- `mosek.iparam.log_cut_second_opt` 290
Controls the reduction in the log levels for the second and any subsequent optimizations.
- `mosek.iparam.log_factor` 291
If turned on, then the factor log lines are added to the log.
- `mosek.iparam.log_feasrepair` 291
Controls the amount of output printed when performing feasibility repair.
- `mosek.iparam.log_file` 291
If turned on, then some log info is printed when a file is written or read.
- `mosek.iparam.log_head` 291
If turned on, then a header line is added to the log.
- `mosek.iparam.log_infeas_ana` 292
Controls log level for the infeasibility analyzer.
- `mosek.iparam.log_intpnt` 292
Controls the amount of log information from the interior-point optimizers.
- `mosek.iparam.log_mio` 292
Controls the amount of log information from the mixed-integer optimizers.
- `mosek.iparam.log_mio_freq` 292
The mixed integer solver logging frequency.
- `mosek.iparam.log_nonconvex` 293
Controls amount of output printed by the nonconvex optimizer.

• <code>mosek.iparam.log_optimizer</code>	293
Controls the amount of general optimizer information that is logged.	
• <code>mosek.iparam.log_order</code>	293
If turned on, then factor lines are added to the log.	
• <code>mosek.iparam.log_param</code>	293
Controls the amount of information printed out about parameter changes.	
• <code>mosek.iparam.log_response</code>	294
Controls amount of output printed when response codes are reported. A higher level implies that more information is logged.	
• <code>mosek.iparam.log_sensitivity</code>	294
Control logging in sensitivity analyzer.	
• <code>mosek.iparam.log_sensitivity_opt</code>	295
Control logging in sensitivity analyzer.	
• <code>mosek.iparam.log_sim</code>	295
Controls the amount of log information from the simplex optimizers.	
• <code>mosek.iparam.log_sim_freq</code>	295
Controls simplex logging frequency.	
• <code>mosek.iparam.log_sim_minor</code>	295
Currently not in use.	
• <code>mosek.iparam.log_sim_network_freq</code>	296
Controls the network simplex logging frequency.	
• <code>mosek.iparam.log_storage</code>	296
Controls the memory related log information.	
• <code>mosek.iparam.max_num_warnings</code>	296
Warning level. A higher value results in more warnings.	
• <code>mosek.iparam.warning_level</code>	326
Warning level.	

14.1.18 Extra information about the optimization problem.

• <code>mosek.iparam.objective_sense</code>	304
If the objective sense for the task is undefined, then the value of this parameter is used as the default objective sense.	

14.1.19 Overall solver parameters.

- `mosek.iparam.bi_clean_optimizer` 276
Controls which simplex optimizer is used in the clean-up phase.
- `mosek.iparam.concurrent_num_optimizers` 279
The maximum number of simultaneous optimizations that will be started by the concurrent optimizer.
- `mosek.iparam.concurrent_priority_dual_simplex` 279
Priority of the dual simplex algorithm when selecting solvers for concurrent optimization.
- `mosek.iparam.concurrent_priority_free_simplex` 279
Priority of the free simplex optimizer when selecting solvers for concurrent optimization.
- `mosek.iparam.concurrent_priority_intpnt` 280
Priority of the interior-point algorithm when selecting solvers for concurrent optimization.
- `mosek.iparam.concurrent_priority_primal_simplex` 280
Priority of the primal simplex algorithm when selecting solvers for concurrent optimization.
- `mosek.iparam.data_check` 281
Enable data checking for debug purposes.
- `mosek.iparam.feasrepair_optimize` 281
Controls which type of feasibility analysis is to be performed.
- `mosek.iparam.infeas_prefer_primal` 282
Controls which certificate is used if both primal- and dual- certificate of infeasibility is available.
- `mosek.iparam.license_wait` 289
Controls if MOSEK should queue for a license if none is available.
- `mosek.iparam.mio_cont_sol` 298
Controls the meaning of interior-point and basic solutions in MIP problems.
- `mosek.iparam.mio_local_branch_number` 300
Controls the size of the local search space when doing local branching.
- `mosek.iparam.mio_mode` 301
Turns on/off the mixed integer mode.
- `mosek.iparam.optimizer` 307
Controls which optimizer is used to optimize the task.
- `mosek.iparam.presolve_level` 309
Currently not used.
- `mosek.iparam.presolve_use` 310
Controls whether the presolve is applied to a problem before it is optimized.

- `mosek.iparam.sensitivity_all` 316
Controls sensitivity report behavior.
- `mosek.iparam.sensitivity_optimizer` 317
Controls which optimizer is used for optimal partition sensitivity analysis.
- `mosek.iparam.sensitivity_type` 317
Controls which type of sensitivity analysis is to be performed.
- `mosek.iparam.solution_callback` 326
Indicates whether solution call-backs will be performed during the optimization.

14.1.20 Behavior of the optimization task.

Parameters defining the behavior of an optimization task when loading data.

- `mosek.iparam.alloc_add_qnz` 276
Controls how the quadratic matrixes are extended.
- `mosek.sparam.feasrepair_name_prefix` 336
Feasibility repair name prefix.
- `mosek.sparam.feasrepair_name_separator` 336
Feasibility repair name separator.
- `mosek.sparam.feasrepair_name_wsumviol` 336
Feasibility repair name violation name.
- `mosek.iparam.maxnumanz_double_trh` 297
Controls how the constraint matrix is extended.
- `mosek.iparam.read_add_anz` 310
Controls how the constraint matrix is extended.
- `mosek.iparam.read_add_con` 310
Additional number of constraints that is made room for in the problem.
- `mosek.iparam.read_add_cone` 311
Additional number of conic constraints that is made room for in the problem.
- `mosek.iparam.read_add_qnz` 311
Controls how the quadratic matrixes are extended.
- `mosek.iparam.read_add_var` 311
Additional number of variables that is made room for in the problem.
- `mosek.iparam.read_anz` 311
Controls the expected number of constraint non-zeros.
- `mosek.iparam.read_con` 311
Controls the expected number of constraints.

- `mosek.iparam.read_cone` 312
Controls the expected number of conic constraints.
- `mosek.iparam.read_qnz` 316
Controls the expected number of quadratic non-zeros.
- `mosek.iparam.read_task_ignore_param` 316
Controls what information is used from the task files.
- `mosek.iparam.read_var` 316
Controls the expected number of variables.
- `mosek.iparam.write_task_inc_sol` 333
Controls whether the solutions are stored in the task file too.

14.1.21 Data input/output parameters.

Parameters defining the behavior of data readers and writers.

- `mosek.sparam.bas_sol_file_name` 335
Name of the bas solution file.
- `mosek.sparam.data_file_name` 335
Data are read and written to this file.
- `mosek.sparam.debug_file_name` 336
MOSEK debug file.
- `mosek.iparam.infeas_report_auto` 282
Turns the feasibility report on or off.
- `mosek.sparam.int_sol_file_name` 337
Name of the int solution file.
- `mosek.sparam.itr_sol_file_name` 337
Name of the itr solution file.
- `mosek.iparam.log_file` 291
If turned on, then some log info is printed when a file is written or read.
- `mosek.iparam.lp_write_ignore_incompatible_items` 296
Controls the result of writing a problem containing incompatible items to an LP file.
- `mosek.iparam.opf_max_terms_per_line` 305
The maximum number of terms (linear and quadratic) per line when an OPF file is written.
- `mosek.iparam.opf_write_header` 305
Write a text header with date and MOSEK version in an OPF file.
- `mosek.iparam.opf_write_hints` 305
Write a hint section with problem dimensions in the beginning of an OPF file.

• <code>mosek.iparam.opf_write_parameters</code>	306
Write a parameter section in an OPF file.	
• <code>mosek.iparam.opf_write_problem</code>	306
Write objective, constraints, bounds etc. to an OPF file.	
• <code>mosek.iparam.opf_write_sol_bas</code>	306
Controls what is written to the OPF files.	
• <code>mosek.iparam.opf_write_sol_itg</code>	306
Controls what is written to the OPF files.	
• <code>mosek.iparam.opf_write_sol_itr</code>	307
Controls what is written to the OPF files.	
• <code>mosek.iparam.opf_write_solutions</code>	307
Enable inclusion of solutions in the OPF files.	
• <code>mosek.sparam.param_comment_sign</code>	337
Solution file comment character.	
• <code>mosek.iparam.param_read_case_name</code>	308
If turned on, then names in the parameter file are case sensitive.	
• <code>mosek.sparam.param_read_file_name</code>	337
Modifications to the parameter database is read from this file.	
• <code>mosek.iparam.param_read_ign_error</code>	308
If turned on, then errors in paramter settings is ignored.	
• <code>mosek.sparam.param_write_file_name</code>	338
The parameter database is written to this file.	
• <code>mosek.iparam.read_add_anz</code>	310
Controls how the constraint matrix is extended.	
• <code>mosek.iparam.read_add_con</code>	310
Additional number of constraints that is made room for in the problem.	
• <code>mosek.iparam.read_add_cone</code>	311
Additional number of conic constraints that is made room for in the problem.	
• <code>mosek.iparam.read_add_qnz</code>	311
Controls how the quadratic matrixes are extended.	
• <code>mosek.iparam.read_add_var</code>	311
Additional number of variables that is made room for in the problem.	
• <code>mosek.iparam.read_anz</code>	311
Controls the expected number of constraint non-zeros.	
• <code>mosek.iparam.read_con</code>	311
Controls the expected number of constraints.	

- `mosek.iparam.read_cone` 312
Controls the expected number of conic constraints.
- `mosek.iparam.read_data_compressed` 312
Controls the input file decompression.
- `mosek.iparam.read_data_format` 312
Format of the data file to be read.
- `mosek.iparam.read_keep_free_con` 313
Controls whether the free constraints are included in the problem.
- `mosek.iparam.read_lp_drop_new_vars_in_bou` 313
Controls how the LP files are interpreted.
- `mosek.iparam.read_lp_quoted_names` 313
If a name is in quotes when reading an LP file, the quotes will be removed.
- `mosek.sparam.read_mps_bou_name` 338
Name of the BOUNDS vector used. An empty name means that the first BOUNDS vector is used.
- `mosek.iparam.read_mps_format` 314
Controls how strictly the MPS file reader interprets the MPS format.
- `mosek.iparam.read_mps_keep_int` 314
Controls if integer constraints are read.
- `mosek.sparam.read_mps_obj_name` 338
Objective name in the MPS file.
- `mosek.iparam.read_mps_obj_sense` 314
Controls the MPS format extensions.
- `mosek.iparam.read_mps_quoted_names` 314
Controls the MPS format extensions.
- `mosek.sparam.read_mps_ran_name` 338
Name of the RANGE vector used. An empty name means that the first RANGE vector is used.
- `mosek.iparam.read_mps_relax` 315
Controls the meaning of integer constraints.
- `mosek.sparam.read_mps_rhs_name` 339
Name of the RHS used. An empty name means that the first RHS vector is used.
- `mosek.iparam.read_mps_width` 315
Controls the maximal number of chars allowed in one line of the MPS file.
- `mosek.iparam.read_q_mode` 315
Controls how the Q matrices are read from the MPS file.

• <code>mosek.iparam.read_qnz</code>	316
Controls the expected number of quadratic non-zeros.	
• <code>mosek.iparam.read_task_ignore_param</code>	316
Controls what information is used from the task files.	
• <code>mosek.iparam.read_var</code>	316
Controls the expected number of variables.	
• <code>mosek.sparam.sensitivity_file_name</code>	339
Sensitivity report file name.	
• <code>mosek.sparam.sensitivity_res_file_name</code>	339
Name of the sensitivity report output file.	
• <code>mosek.sparam.sol_filter_xc_low</code>	339
Solution file filter.	
• <code>mosek.sparam.sol_filter_xc_upr</code>	340
Solution file filter.	
• <code>mosek.sparam.sol_filter_xx_low</code>	340
Solution file filter.	
• <code>mosek.sparam.sol_filter_xx_upr</code>	340
Solution file filter.	
• <code>mosek.iparam.sol_quoted_names</code>	325
Controls the solution file format.	
• <code>mosek.iparam.sol_read_name_width</code>	325
Controls the input solution file format.	
• <code>mosek.iparam.sol_read_width</code>	325
Controls the input solution file format.	
• <code>mosek.sparam.stat_file_name</code>	341
Statistics file name.	
• <code>mosek.sparam.stat_key</code>	341
Key used when writing the summary file.	
• <code>mosek.sparam.stat_name</code>	341
Name used when writing the statistics file.	
• <code>mosek.iparam.write_bas_constraints</code>	326
Controls the basic solution file format.	
• <code>mosek.iparam.write_bas_head</code>	327
Controls the basic solution file format.	
• <code>mosek.iparam.write_bas_variables</code>	327
Controls the basic solution file format.	

• <code>mosek.iparam.write_data_compressed</code>	327
Controls output file compression.	
• <code>mosek.iparam.write_data_format</code>	327
Controls the output file problem format.	
• <code>mosek.iparam.write_data_param</code>	328
Controls output file data.	
• <code>mosek.iparam.write_free_con</code>	328
Controls the output file data.	
• <code>mosek.iparam.write_generic_names</code>	328
Controls the output file data.	
• <code>mosek.iparam.write_generic_names_io</code>	329
Index origin used in generic names.	
• <code>mosek.iparam.write_int_constraints</code>	329
Controls the integer solution file format.	
• <code>mosek.iparam.write_int_head</code>	329
Controls the integer solution file format.	
• <code>mosek.iparam.write_int_variables</code>	329
Controls the integer solution file format.	
• <code>mosek.sparam.write_lp_gen_var_name</code>	341
Added variable names in the LP files.	
• <code>mosek.iparam.write_lp_line_width</code>	330
Controls the LP output file format.	
• <code>mosek.iparam.write_lp_quoted_names</code>	330
Controls LP output file format.	
• <code>mosek.iparam.write_lp_strict_format</code>	330
Controls whether LP output files satisfy the LP format strictly.	
• <code>mosek.iparam.write_lp_terms_per_line</code>	330
Controls the LP output file format.	
• <code>mosek.iparam.write_mps_int</code>	331
Controls the output file data.	
• <code>mosek.iparam.write_mps_obj_sense</code>	331
Controls the output file data.	
• <code>mosek.iparam.write_mps_quoted_names</code>	331
Controls the output file data.	
• <code>mosek.iparam.write_mps_strict</code>	332
Controls the output MPS file format.	

- `mosek.iparam.write_precision` 332
Controls data precision employed in when writing an MPS file.
- `mosek.iparam.write_sol_constraints` 332
Controls the solution file format.
- `mosek.iparam.write_sol_head` 332
Controls solution file format.
- `mosek.iparam.write_sol_variables` 333
Controls the solution file format.
- `mosek.iparam.write_task_inc_sol` 333
Controls whether the solutions are stored in the task file too.
- `mosek.iparam.write_xml_mode` 333
Controls if linear coefficients should be written by row or column when writing in the XML file format.

14.1.22 Solution input/output parameters.

Parameters defining the behavior of solution reader and writer.

- `mosek.sparam.bas_sol_file_name` 335
Name of the bas solution file.
- `mosek.iparam.infeas_report_auto` 282
Turns the feasibility report on or off.
- `mosek.sparam.int_sol_file_name` 337
Name of the int solution file.
- `mosek.sparam.itr_sol_file_name` 337
Name of the itr solution file.
- `mosek.iparam.sol_filter_keep_basic` 324
Controls the license manager client behavior.
- `mosek.iparam.sol_filter_keep_ranged` 325
Control the contents of the solution files.
- `mosek.sparam.sol_filter_xc_low` 339
Solution file filter.
- `mosek.sparam.sol_filter_xc_upr` 340
Solution file filter.
- `mosek.sparam.sol_filter_xx_low` 340
Solution file filter.

• <code>mosek.sparam.sol_filter_xx_upr</code>	340
Solution file filter.	
• <code>mosek.iparam.sol_quoted_names</code>	325
Controls the solution file format.	
• <code>mosek.iparam.sol_read_name_width</code>	325
Controls the input solution file format.	
• <code>mosek.iparam.sol_read_width</code>	325
Controls the input solution file format.	
• <code>mosek.iparam.write_bas_constraints</code>	326
Controls the basic solution file format.	
• <code>mosek.iparam.write_bas_head</code>	327
Controls the basic solution file format.	
• <code>mosek.iparam.write_bas_variables</code>	327
Controls the basic solution file format.	
• <code>mosek.iparam.write_int_constraints</code>	329
Controls the integer solution file format.	
• <code>mosek.iparam.write_int_head</code>	329
Controls the integer solution file format.	
• <code>mosek.iparam.write_int_variables</code>	329
Controls the integer solution file format.	
• <code>mosek.iparam.write_sol_constraints</code>	332
Controls the solution file format.	
• <code>mosek.iparam.write_sol_head</code>	332
Controls solution file format.	
• <code>mosek.iparam.write_sol_variables</code>	333
Controls the solution file format.	

14.1.23 Infeasibility report parameters.

• <code>mosek.iparam.infeas_generic_names</code>	282
Controls the contents of the infeasibility report.	
• <code>mosek.iparam.infeas_report_level</code>	282
Controls the contents of the infeasibility report.	
• <code>mosek.iparam.log_infeas_ana</code>	292
Controls log level for the infeasibility analyzer.	

14.1.24 License manager parameters.

- `mosek.iparam.license_allow_overuse` 287
Controls if license overuse is allowed when caching licenses
- `mosek.iparam.license_cache_time` 287
Controls the license manager client behavior.
- `mosek.iparam.license_check_time` 288
Controls the license manager client behavior.
- `mosek.iparam.license_debug` 288
Controls the license manager client debugging behavior.
- `mosek.iparam.license_pause_time` 288
Controls license manager client behavior.
- `mosek.iparam.license_suppress_expire_wrns` 288
Controls license manager client behavior.
- `mosek.iparam.license_wait` 289
Controls if MOSEK should queue for a license if none is available.

14.1.25 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` 278
Specify the level of convexity check on quadratic problems
- `mosek.iparam.check_task_data` 279
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` 250
Data tolerance threshold.
- `mosek.dparam.data_tol_aij_large` 250
Data tolerance threshold.
- `mosek.dparam.data_tol_bound_inf` 250
Data tolerance threshold.
- `mosek.dparam.data_tol_bound_wrn` 250
Data tolerance threshold.
- `mosek.dparam.data_tol_c_huge` 251
Data tolerance threshold.

- `mosek.dparam.data_tol_cj_large` 251
Data tolerance threshold.
- `mosek.dparam.data_tol_qij` 251
Data tolerance threshold.
- `mosek.dparam.data_tol_x` 251
Data tolerance threshold.

14.2 Double parameters

- `mosek.dparam.basis_rel_tol_s` 248
Maximum relative dual bound violation allowed in an optimal basic solution.
- `mosek.dparam.basis_tol_s` 249
Maximum absolute dual bound violation in an optimal basic solution.
- `mosek.dparam.basis_tol_x` 249
Maximum absolute primal bound violation allowed in an optimal basic solution.
- `mosek.dparam.bi_lu_tol_rel_piv` 249
Relative pivot tolerance used in the LU factorization in the basis identification procedure.
- `mosek.dparam.callback_freq` 249
Controls progress call-back frequency.
- `mosek.dparam.data_tol_aij` 250
Data tolerance threshold.
- `mosek.dparam.data_tol_aij_large` 250
Data tolerance threshold.
- `mosek.dparam.data_tol_bound_inf` 250
Data tolerance threshold.
- `mosek.dparam.data_tol_bound_wrn` 250
Data tolerance threshold.
- `mosek.dparam.data_tol_c_huge` 251
Data tolerance threshold.
- `mosek.dparam.data_tol_cj_large` 251
Data tolerance threshold.
- `mosek.dparam.data_tol_qij` 251
Data tolerance threshold.
- `mosek.dparam.data_tol_x` 251
Data tolerance threshold.

- `mosek.dparam.feasrepair_tol` 252
Tolerance for constraint enforcing upper bound on sum of weighted violations in feasibility repair.
- `mosek.dparam.intpnt_co_tol_dfeas` 252
Dual feasibility tolerance used by the conic interior-point optimizer.
- `mosek.dparam.intpnt_co_tol_infeas` 252
Infeasibility tolerance for the conic solver.
- `mosek.dparam.intpnt_co_tol_mu_red` 252
Optimality tolerance for the conic solver.
- `mosek.dparam.intpnt_co_tol_near_rel` 253
Optimality tolerance for the conic solver.
- `mosek.dparam.intpnt_co_tol_pfeas` 253
Primal feasibility tolerance used by the conic interior-point optimizer.
- `mosek.dparam.intpnt_co_tol_rel_gap` 253
Relative gap termination tolerance used by the conic interior-point optimizer.
- `mosek.dparam.intpnt_nl_merit_bal` 253
Controls if the complementarity and infeasibility is converging to zero at about equal rates.
- `mosek.dparam.intpnt_nl_tol_dfeas` 254
Dual feasibility tolerance used when a nonlinear model is solved.
- `mosek.dparam.intpnt_nl_tol_mu_red` 254
Relative complementarity gap tolerance.
- `mosek.dparam.intpnt_nl_tol_near_rel` 254
Nonlinear solver optimality tolerance parameter.
- `mosek.dparam.intpnt_nl_tol_pfeas` 255
Primal feasibility tolerance used when a nonlinear model is solved.
- `mosek.dparam.intpnt_nl_tol_rel_gap` 255
Relative gap termination tolerance for nonlinear problems.
- `mosek.dparam.intpnt_nl_tol_rel_step` 255
Relative step size to the boundary for general nonlinear optimization problems.
- `mosek.dparam.intpnt_tol_dfeas` 255
Dual feasibility tolerance used for linear and quadratic optimization problems.
- `mosek.dparam.intpnt_tol_dsafe` 255
Controls the interior-point dual starting point.
- `mosek.dparam.intpnt_tol_infeas` 256
Nonlinear solver infeasibility tolerance parameter.
- `mosek.dparam.intpnt_tol_mu_red` 256
Relative complementarity gap tolerance.

• <code>mosek.dparam.intpnt_tol_path</code>	256
interior-point centering aggressiveness.	
• <code>mosek.dparam.intpnt_tol_pfeas</code>	257
Primal feasibility tolerance used for linear and quadratic optimization problems.	
• <code>mosek.dparam.intpnt_tol_psafe</code>	257
Controls the interior-point primal starting point.	
• <code>mosek.dparam.intpnt_tol_rel_gap</code>	257
Relative gap termination tolerance.	
• <code>mosek.dparam.intpnt_tol_rel_step</code>	257
Relative step size to the boundary for linear and quadratic optimization problems.	
• <code>mosek.dparam.intpnt_tol_step_size</code>	258
If the step size falls below the value of this parameter, then the interior-point optimizer assumes it is stalled. If it does not make any progress.	
• <code>mosek.dparam.lower_obj_cut</code>	258
Objective bound.	
• <code>mosek.dparam.lower_obj_cut_finite_trh</code>	258
Objective bound.	
• <code>mosek.dparam.mio_disable_term_time</code>	258
Certain termination criterias is disabled within the mixed integer optimizer for period time specified by the parameter.	
• <code>mosek.dparam.mio_heuristic_time</code>	259
Time limit for the mixed integer heuristics.	
• <code>mosek.dparam.mio_max_time</code>	259
Time limit for the mixed integer optimizer.	
• <code>mosek.dparam.mio_max_time_aprx_opt</code>	259
Time limit for the mixed integer optimizer.	
• <code>mosek.dparam.mio_near_tol_abs_gap</code>	260
Relaxed absolute optimality tolerance employed by the mixed integer optimizer.	
• <code>mosek.dparam.mio_near_tol_rel_gap</code>	260
The mixed integer optimizer is terminated when this tolerance is satisfied.	
• <code>mosek.dparam.mio_rel_add_cut_limited</code>	260
Controls cut generation for MIP solver.	
• <code>mosek.dparam.mio_tol_abs_gap</code>	261
Absolute optimality tolerance employed by the mixed integer optimizer.	
• <code>mosek.dparam.mio_tol_abs_relax_int</code>	261
Integer constraint tolerance.	

- `mosek.dparam.mio_tol_rel_gap` 261
Relative optimality tolerance employed by the mixed integer optimizer.
- `mosek.dparam.mio_tol_rel_relax_int` 261
Integer constraint tolerance.
- `mosek.dparam.mio_tol_x` 262
Absolute solution tolerance used in mixed-integer optimizer.
- `mosek.dparam.nonconvex_tol_feas` 262
Feasibility tolerance used by the nonconvex optimizer.
- `mosek.dparam.nonconvex_tol_opt` 262
Optimality tolerance used by the nonconvex optimizer.
- `mosek.dparam.optimizer_max_time` 262
Solver time limit.
- `mosek.dparam.presolve_tol_aij` 263
Absolute zero tolerance employed for constraint coefficients in the presolve.
- `mosek.dparam.presolve_tol_lin_dep` 263
Controls when a constraint is determined to be linearly dependent.
- `mosek.dparam.presolve_tol_s` 263
Absolute zero tolerance employed for slack variables in the presolve.
- `mosek.dparam.presolve_tol_x` 263
Absolute zero tolerance employed for variables in the presolve.
- `mosek.dparam.simplex_abs_tol_piv` 264
Absolute pivot tolerance employed by the simplex optimizers.
- `mosek.dparam.upper_obj_cut` 264
Objective bound.
- `mosek.dparam.upper_obj_cut_finite_trh` 264
Objective bound.
- `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

- `bi_lu_tol_rel_piv`

Corresponding constant:`mosek.dparam.bi_lu_tol_rel_piv`**Description:**

Relative pivot tolerance used in the LU factorization in the basis identification procedure.

Possible Values:

Any number between 1.0e-6 and 0.999999.

Default value:

0.01

- `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

• **data_tol_ajj****Corresponding constant:**`mosek.dparam.data_tol_ajj`**Description:**Absolute zero tolerance for elements in A .**Possible Values:**Any number between $1.0\text{e-}16$ and $1.0\text{e-}6$.**Default value:** $1.0\text{e-}12$ • **data_tol_ajj_large****Corresponding constant:**`mosek.dparam.data_tol_ajj_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 $+\text{inf}$.**Default value:** $1.0\text{e}10$ • **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 $+\text{inf}$.**Default value:** $1.0\text{e}16$ • **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 $+\text{inf}$.

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 $+\infty$.

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 $+\infty$.

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 $+\infty$.

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 +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 1.0e-16 and 1.0e+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

- `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

- `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

- `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 +inf.

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 might 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 might 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 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 a feasible solution having an objective value 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

- `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 criterias are never checked.

Possible Values:

Any number between 0.0 and +inf.

Default value:

0.0

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 -inf and +inf.

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 -inf and +inf.

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 criterias 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-5

See also:

`mosek.dparam.mio_disable_term_time` Certain termination criterias 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_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 $+\text{inf}$.

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 $+\text{inf}$.

Default value:

1.0e-5

- `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 $+\text{inf}$.

Default value:

1.0e-8

- `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 $-\infty$ and $+\infty$.

Default value:

-1.0

- `presolve_tol_aj`

Corresponding constant:

`mosek.dparam.presolve_tol_aj`

Description:

Absolute zero tolerance employed for a_{ij} in the presolve.

Possible Values:

Any number between 0.0 and $+\infty$.

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 $+\infty$.

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 $+\infty$.

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

- `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 a feasible solution having an objective value 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

- `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 upper objective cut `mosek.dparam.upper_obj_cut` is treated as ∞ .

Possible Values:

Any number between -inf and +inf.

Default value:

0.5e30

14.3 Integer parameters

- `mosek.iparam.alloc_add_qnz` 276
Controls how the quadratic matrixes are extended.
- `mosek.iparam.bi_clean_optimizer` 276
Controls which simplex optimizer is used in the clean-up phase.
- `mosek.iparam.bi_ignore_max_iter` 276
Turns on basis identification in case the interior-point optimizer is terminated due to maximum number of iterations.
- `mosek.iparam.bi_ignore_num_error` 277
Turns on basis identification in case the interior-point optimizer is terminated due to a numerical problem.
- `mosek.iparam.bi_max_iterations` 277
Maximum number of iterations after basis identification.
- `mosek.iparam.cache_size_l1` 277
Specifies the size of the level 1 cache of the processor.
- `mosek.iparam.cache_size_l2` 278
Specifies the size of the level 2 cache of the processor.
- `mosek.iparam.check_convexity` 278
Specify the level of convexity check on quadratic problems
- `mosek.iparam.check_ctrl_c` 278
Turns ctrl-c check on or off.
- `mosek.iparam.check_task_data` 279
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` 279
The maximum number of simultaneous optimizations that will be started by the concurrent optimizer.
- `mosek.iparam.concurrent_priority_dual_simplex` 279
Priority of the dual simplex algorithm when selecting solvers for concurrent optimization.
- `mosek.iparam.concurrent_priority_free_simplex` 279
Priority of the free simplex optimizer when selecting solvers for concurrent optimization.
- `mosek.iparam.concurrent_priority_intpnt` 280
Priority of the interior-point algorithm when selecting solvers for concurrent optimization.
- `mosek.iparam.concurrent_priority_primal_simplex` 280
Priority of the primal simplex algorithm when selecting solvers for concurrent optimization.

• <code>mosek.iparam.cpu_type</code>	280
Specifies the CPU type.	
• <code>mosek.iparam.data_check</code>	281
Enable data checking for debug purposes.	
• <code>mosek.iparam.feasrepair_optimize</code>	281
Controls which type of feasibility analysis is to be performed.	
• <code>mosek.iparam.flush_stream_freq</code>	281
Controls the stream flushing frequency.	
• <code>mosek.iparam.infeas_generic_names</code>	282
Controls the contents of the infeasibility report.	
• <code>mosek.iparam.infeas_prefer_primal</code>	282
Controls which certificate is used if both primal- and dual- certificate of infeasibility is available.	
• <code>mosek.iparam.infeas_report_auto</code>	282
Turns the feasibility report on or off.	
• <code>mosek.iparam.infeas_report_level</code>	282
Controls the contents of the infeasibility report.	
• <code>mosek.iparam.intpnt_basis</code>	283
Controls whether basis identification is performed.	
• <code>mosek.iparam.intpnt_diff_step</code>	283
Controls whether different step sizes are allowed in the primal and dual space.	
• <code>mosek.iparam.intpnt_factor_debug_lvl</code>	284
Controls factorization debug level.	
• <code>mosek.iparam.intpnt_factor_method</code>	284
Controls the method used to factor the Newton equation system.	
• <code>mosek.iparam.intpnt_max_iterations</code>	284
Controls the maximum number of iterations allowed in the interior-point optimizer.	
• <code>mosek.iparam.intpnt_max_num_cor</code>	284
Maximum number of correction steps.	
• <code>mosek.iparam.intpnt_max_num_refinement_steps</code>	284
Maximum number of steps to be used by the iterative search direction refinement.	
• <code>mosek.iparam.intpnt_num_threads</code>	285
Controls the number of threads employed by the interior-point optimizer.	
• <code>mosek.iparam.intpnt_off_col_trh</code>	285
Controls the aggressiveness of the offending column detection.	
• <code>mosek.iparam.intpnt_order_method</code>	285
Controls the ordering strategy.	

- `mosek.iparam.intpnt_regularization_use` 286
Controls whether regularization is allowed.
- `mosek.iparam.intpnt_scaling` 286
Controls how the problem is scaled before the interior-point optimizer is used.
- `mosek.iparam.intpnt_solve_form` 286
Controls whether the primal or the dual problem is solved.
- `mosek.iparam.intpnt_starting_point` 287
Starting point used by the interior-point optimizer.
- `mosek.iparam.license_allow_overuse` 287
Controls if license overuse is allowed when caching licenses
- `mosek.iparam.license_cache_time` 287
Controls the license manager client behavior.
- `mosek.iparam.license_check_time` 288
Controls the license manager client behavior.
- `mosek.iparam.license_debug` 288
Controls the license manager client debugging behavior.
- `mosek.iparam.license_pause_time` 288
Controls license manager client behavior.
- `mosek.iparam.license_suppress_expire_wrns` 288
Controls license manager client behavior.
- `mosek.iparam.license_wait` 289
Controls if MOSEK should queue for a license if none is available.
- `mosek.iparam.log` 289
Controls the amount of log information.
- `mosek.iparam.log_bi` 289
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` 290
Controls the logging frequency.
- `mosek.iparam.log_concurrent` 290
Controls amount of output printed by the concurrent optimizer.
- `mosek.iparam.log_cut_second_opt` 290
Controls the reduction in the log levels for the second and any subsequent optimizations.
- `mosek.iparam.log_factor` 291
If turned on, then the factor log lines are added to the log.

• <code>mosek.iparam.log_feasrepair</code>	291
Controls the amount of output printed when performing feasibility repair.	
• <code>mosek.iparam.log_file</code>	291
If turned on, then some log info is printed when a file is written or read.	
• <code>mosek.iparam.log_head</code>	291
If turned on, then a header line is added to the log.	
• <code>mosek.iparam.log_infeas_ana</code>	292
Controls log level for the infeasibility analyzer.	
• <code>mosek.iparam.log_intpnt</code>	292
Controls the amount of log information from the interior-point optimizers.	
• <code>mosek.iparam.log_mio</code>	292
Controls the amount of log information from the mixed-integer optimizers.	
• <code>mosek.iparam.log_mio_freq</code>	292
The mixed integer solver logging frequency.	
• <code>mosek.iparam.log_nonconvex</code>	293
Controls amount of output printed by the nonconvex optimizer.	
• <code>mosek.iparam.log_optimizer</code>	293
Controls the amount of general optimizer information that is logged.	
• <code>mosek.iparam.log_order</code>	293
If turned on, then factor lines are added to the log.	
• <code>mosek.iparam.log_param</code>	293
Controls the amount of information printed out about parameter changes.	
• <code>mosek.iparam.log_presolve</code>	294
Controls amount of output printed by the presolve procedure. A higher level implies that more information is logged.	
• <code>mosek.iparam.log_response</code>	294
Controls amount of output printed when response codes are reported. A higher level implies that more information is logged.	
• <code>mosek.iparam.log_sensitivity</code>	294
Control logging in sensitivity analyzer.	
• <code>mosek.iparam.log_sensitivity_opt</code>	295
Control logging in sensitivity analyzer.	
• <code>mosek.iparam.log_sim</code>	295
Controls the amount of log information from the simplex optimizers.	
• <code>mosek.iparam.log_sim_freq</code>	295
Controls simplex logging frequency.	

• <code>mosek.iparam.log_sim_minor</code>	295
Currently not in use.	
• <code>mosek.iparam.log_sim_network_freq</code>	296
Controls the network simplex logging frequency.	
• <code>mosek.iparam.log_storage</code>	296
Controls the memory related log information.	
• <code>mosek.iparam.lp_write_ignore_incompatible_items</code>	296
Controls the result of writing a problem containing incompatible items to an LP file.	
• <code>mosek.iparam.max_num_warnings</code>	296
Warning level. A higher value results in more warnings.	
• <code>mosek.iparam.maxnumanz_double_trh</code>	297
Controls how the constraint matrix is extended.	
• <code>mosek.iparam.mio_branch_dir</code>	297
Controls whether the mixed integer optimizer is branching up or down by default.	
• <code>mosek.iparam.mio_branch_priorities_use</code>	297
Controls whether branching priorities are used by the mixed integer optimizer.	
• <code>mosek.iparam.mio_construct_sol</code>	297
Controls if initial MIP solution should be constructed from value of integer variables.	
• <code>mosek.iparam.mio_cont_sol</code>	298
Controls the meaning of interior-point and basic solutions in MIP problems.	
• <code>mosek.iparam.mio_cut_level_root</code>	298
Controls the cut level employed by the mixed integer optimizer at the root node.	
• <code>mosek.iparam.mio_cut_level_tree</code>	299
Controls the cut level employed by the mixed integer optimizer in the tree.	
• <code>mosek.iparam.mio_feaspump_level</code>	299
Controls the feasibility pump heuristic which is used to construct a good initial feasible solution.	
• <code>mosek.iparam.mio_heuristic_level</code>	299
Controls the heuristic employed by the mixed integer optimizer to locate an initial integer feasible solution.	
• <code>mosek.iparam.mio_keep_basis</code>	300
Controls whether the integer presolve keeps bases in memory.	
• <code>mosek.iparam.mio_local_branch_number</code>	300
Controls the size of the local search space when doing local branching.	
• <code>mosek.iparam.mio_max_num_branches</code>	300
Maximum number of branches allowed during the branch and bound search.	

- `mosek.iparam.mio_max_num_relaxs` 301
Maximum number of relaxations in branch and bound search.
- `mosek.iparam.mio_max_num_solutions` 301
Controls how many feasible solutions the mixed-integer optimizer investigates.
- `mosek.iparam.mio_mode` 301
Turns on/off the mixed integer mode.
- `mosek.iparam.mio_node_optimizer` 302
Controls which optimizer is employed at the non-root nodes in the mixed integer optimizer.
- `mosek.iparam.mio_node_selection` 302
Controls the node selection strategy employed by the mixed integer optimizer.
- `mosek.iparam.mio_presolve_aggregate` 303
Controls whether problem aggregation is performed in the mixed integer presolve.
- `mosek.iparam.mio_presolve_probing` 303
Controls whether probing is employed by the mixed integer presolve.
- `mosek.iparam.mio_presolve_use` 303
Controls whether presolve is performed by the mixed integer optimizer.
- `mosek.iparam.mio_root_optimizer` 303
Controls which optimizer is employed at the root node in the mixed integer optimizer.
- `mosek.iparam.mio_strong_branch` 304
The depth from the root in which strong branching is employed.
- `mosek.iparam.nonconvex_max_iterations` 304
Maximum number of iterations that can be used by the nonconvex optimizer.
- `mosek.iparam.objective_sense` 304
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` 305
The maximum number of terms (linear and quadratic) per line when an OPF file is written.
- `mosek.iparam.opf_write_header` 305
Write a text header with date and MOSEK version in an OPF file.
- `mosek.iparam.opf_write_hints` 305
Write a hint section with problem dimensions in the beginning of an OPF file.
- `mosek.iparam.opf_write_parameters` 306
Write a parameter section in an OPF file.
- `mosek.iparam.opf_write_problem` 306
Write objective, constraints, bounds etc. to an OPF file.

• <code>mosek.iparam.opf_write_sol_bas</code>	306
Controls what is written to the OPF files.	
• <code>mosek.iparam.opf_write_sol_itg</code>	306
Controls what is written to the OPF files.	
• <code>mosek.iparam.opf_write_sol_itr</code>	307
Controls what is written to the OPF files.	
• <code>mosek.iparam.opf_write_solutions</code>	307
Enable inclusion of solutions in the OPF files.	
• <code>mosek.iparam.optimizer</code>	307
Controls which optimizer is used to optimize the task.	
• <code>mosek.iparam.param_read_case_name</code>	308
If turned on, then names in the parameter file are case sensitive.	
• <code>mosek.iparam.param_read_ign_error</code>	308
If turned on, then errors in paramter settings is ignored.	
• <code>mosek.iparam.presolve_elim_fill</code>	308
Maximum amount of fill-in in the elimination phase.	
• <code>mosek.iparam.presolve_eliminator_use</code>	309
Controls whether free or implied free variables are eliminated from the problem.	
• <code>mosek.iparam.presolve_level</code>	309
Currently not used.	
• <code>mosek.iparam.presolve_lindep_use</code>	309
Controls whether the linear constraints are checked for linear dependencies.	
• <code>mosek.iparam.presolve_lindep_work_lim</code>	309
Controls linear dependency check in presolve.	
• <code>mosek.iparam.presolve_use</code>	310
Controls whether the presolve is applied to a problem before it is optimized.	
• <code>mosek.iparam.read_add_anz</code>	310
Controls how the constraint matrix is extended.	
• <code>mosek.iparam.read_add_con</code>	310
Additional number of constraints that is made room for in the problem.	
• <code>mosek.iparam.read_add_cone</code>	311
Additional number of conic constraints that is made room for in the problem.	
• <code>mosek.iparam.read_add_qnz</code>	311
Controls how the quadratic matrixes are extended.	
• <code>mosek.iparam.read_add_var</code>	311
Additional number of variables that is made room for in the problem.	

• <code>mosek.iparam.read_anz</code>	311
Controls the expected number of constraint non-zeros.	
• <code>mosek.iparam.read_con</code>	311
Controls the expected number of constraints.	
• <code>mosek.iparam.read_cone</code>	312
Controls the expected number of conic constraints.	
• <code>mosek.iparam.read_data_compressed</code>	312
Controls the input file decompression.	
• <code>mosek.iparam.read_data_format</code>	312
Format of the data file to be read.	
• <code>mosek.iparam.read_keep_free_con</code>	313
Controls whether the free constraints are included in the problem.	
• <code>mosek.iparam.read_lp_drop_new_vars_in_bou</code>	313
Controls how the LP files are interpreted.	
• <code>mosek.iparam.read_lp_quoted_names</code>	313
If a name is in quotes when reading an LP file, the quotes will be removed.	
• <code>mosek.iparam.read_mps_format</code>	314
Controls how strictly the MPS file reader interprets the MPS format.	
• <code>mosek.iparam.read_mps_keep_int</code>	314
Controls if integer constraints are read.	
• <code>mosek.iparam.read_mps_obj_sense</code>	314
Controls the MPS format extensions.	
• <code>mosek.iparam.read_mps_quoted_names</code>	314
Controls the MPS format extensions.	
• <code>mosek.iparam.read_mps_relax</code>	315
Controls the meaning of integer constraints.	
• <code>mosek.iparam.read_mps_width</code>	315
Controls the maximal number of chars allowed in one line of the MPS file.	
• <code>mosek.iparam.read_q_mode</code>	315
Controls how the Q matrices are read from the MPS file.	
• <code>mosek.iparam.read_qnz</code>	316
Controls the expected number of quadratic non-zeros.	
• <code>mosek.iparam.read_task_ignore_param</code>	316
Controls what information is used from the task files.	
• <code>mosek.iparam.read_var</code>	316
Controls the expected number of variables.	

• <code>mosek.iparam.sensitivity_all</code>	316
Controls sensitivity report behavior.	
• <code>mosek.iparam.sensitivity_optimizer</code>	317
Controls which optimizer is used for optimal partition sensitivity analysis.	
• <code>mosek.iparam.sensitivity_type</code>	317
Controls which type of sensitivity analysis is to be performed.	
• <code>mosek.iparam.sim_degen</code>	318
Controls how aggressive degeneration is approached.	
• <code>mosek.iparam.sim_dual_crash</code>	318
Controls whether crashing is performed in the dual simplex optimizer.	
• <code>mosek.iparam.sim_dual_restrict_selection</code>	318
Controls how aggressively restricted selection is used.	
• <code>mosek.iparam.sim_dual_selection</code>	319
Controls the dual simplex strategy.	
• <code>mosek.iparam.sim_hotstart</code>	319
Controls the type of hot-start that the simplex optimizer perform.	
• <code>mosek.iparam.sim_max_iterations</code>	319
Maximum number of iterations that can be used by a simplex optimizer.	
• <code>mosek.iparam.sim_max_num_setbacks</code>	320
Controls how many setbacks that are allowed within a simplex optimizer.	
• <code>mosek.iparam.sim_network_detect</code>	320
Level of aggressiveness of network detection.	
• <code>mosek.iparam.sim_network_detect_hotstart</code>	320
Level of aggressiveness of network detection in a simplex hot-start.	
• <code>mosek.iparam.sim_network_detect_method</code>	321
Controls which type of detection method the network extraction should use.	
• <code>mosek.iparam.sim_non_singular</code>	321
Controls if the simplex optimizer ensures a non-singular basis, if possible.	
• <code>mosek.iparam.sim_primal_crash</code>	321
Controls the simplex crash.	
• <code>mosek.iparam.sim_primal_restrict_selection</code>	322
Controls how aggressively restricted selection is used.	
• <code>mosek.iparam.sim_primal_selection</code>	322
Controls the primal simplex strategy.	
• <code>mosek.iparam.sim_refactor_freq</code>	322
Controls the basis refactoring frequency.	

- `mosek.iparam.sim_save_lu` 323
Controls if the LU factorization stored should be replaced with the LU factorization corresponding to the initial basis.
- `mosek.iparam.sim_scaling` 323
Controls how the problem is scaled before a simplex optimizer is used.
- `mosek.iparam.sim_solve_form` 323
Controls whether the primal or the dual problem is solved by the primal-/dual- simplex optimizer.
- `mosek.iparam.sim_stability_priority` 324
Controls how high priority the numerical stability should be given.
- `mosek.iparam.sim_switch_optimizer` 324
Controls the simplex behavior.
- `mosek.iparam.sol_filter_keep_basic` 324
Controls the license manager client behavior.
- `mosek.iparam.sol_filter_keep_ranged` 325
Control the contents of the solution files.
- `mosek.iparam.sol_quoted_names` 325
Controls the solution file format.
- `mosek.iparam.sol_read_name_width` 325
Controls the input solution file format.
- `mosek.iparam.sol_read_width` 325
Controls the input solution file format.
- `mosek.iparam.solution_callback` 326
Indicates whether solution call-backs will be performed during the optimization.
- `mosek.iparam.warning_level` 326
Warning level.
- `mosek.iparam.write_bas_constraints` 326
Controls the basic solution file format.
- `mosek.iparam.write_bas_head` 327
Controls the basic solution file format.
- `mosek.iparam.write_bas_variables` 327
Controls the basic solution file format.
- `mosek.iparam.write_data_compressed` 327
Controls output file compression.
- `mosek.iparam.write_data_format` 327
Controls the output file problem format.

• <code>mosek.iparam.write_data_param</code>	328
Controls output file data.	
• <code>mosek.iparam.write_free_con</code>	328
Controls the output file data.	
• <code>mosek.iparam.write_generic_names</code>	328
Controls the output file data.	
• <code>mosek.iparam.write_generic_names_io</code>	329
Index origin used in generic names.	
• <code>mosek.iparam.write_int_constraints</code>	329
Controls the integer solution file format.	
• <code>mosek.iparam.write_int_head</code>	329
Controls the integer solution file format.	
• <code>mosek.iparam.write_int_variables</code>	329
Controls the integer solution file format.	
• <code>mosek.iparam.write_lp_line_width</code>	330
Controls the LP output file format.	
• <code>mosek.iparam.write_lp_quoted_names</code>	330
Controls LP output file format.	
• <code>mosek.iparam.write_lp_strict_format</code>	330
Controls whether LP output files satisfy the LP format strictly.	
• <code>mosek.iparam.write_lp_terms_per_line</code>	330
Controls the LP output file format.	
• <code>mosek.iparam.write_mps_int</code>	331
Controls the output file data.	
• <code>mosek.iparam.write_mps_obj_sense</code>	331
Controls the output file data.	
• <code>mosek.iparam.write_mps_quoted_names</code>	331
Controls the output file data.	
• <code>mosek.iparam.write_mps_strict</code>	332
Controls the output MPS file format.	
• <code>mosek.iparam.write_precision</code>	332
Controls data precision employed in when writing an MPS file.	
• <code>mosek.iparam.write_sol_constraints</code>	332
Controls the solution file format.	
• <code>mosek.iparam.write_sol_head</code>	332
Controls solution file format.	

- `mosek.iparam.write_sol_variables` 333
Controls the solution file format.
- `mosek.iparam.write_task_inc_sol` 333
Controls whether the solutions are stored in the task file too.
- `mosek.iparam.write_xml_mode` 333
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 $+\infty$.

Default value:

5000

- `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.conic` Another cone optimizer.
`mosek.optimizertype.nonconvex` The optimizer for nonconvex nonlinear problems.
`mosek.optimizertype.qcone` The Qcone optimizer is used.
`mosek.optimizertype.primal_simplex` The primal simplex optimizer is used.
`mosek.optimizertype.free_simplex` Either the primal or the dual simplex optimizer 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_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 -inf and +inf.

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 -inf and +inf.

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.

Default value:

`mosek.checkconvexitytype.simple`

- `check_ctrl_c`

Corresponding constant:

`mosek.iparam.check_ctrl_c`

Description:

Specifies whether MOSEK should check for <ctrl>+<c> key presses. In case it has, then control is returned to the user program.

In case a user-defined ctrl-c function is defined then that is used to check for ctrl-c. Otherwise the system procedure `signal` is used.

Possible Values:

`mosek.onoffkey.on` Switch the option on.
`mosek.onoffkey.off` Switch the option off.

Default value:

`mosek.onoffkey.off`

- `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.off`

- `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`

- `flush_stream_freq`

Corresponding constant:

`mosek.iparam.flush_stream_freq`

Description:

Controls how frequent the message and log streams are flushed. A value of 0 means that it is never flushed. Otherwise a larger value results in less frequent flushes.

Possible Values:

Any number between 0 and $+\infty$.

Default value:

24

• **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 +inf.

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 $+\infty$.

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 $+\infty$.

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 $+\infty$.

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 $+\infty$.

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.

Possible Values:

Any integer greater than 1.

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.constant` The starting point is set to a constant. This is more reliable than a non-constant starting point.

`mosek.startpointtype.free` The starting point is chosen automatically.

Default value:

`mosek.startpointtype.free`

- `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:

Controls the amount of time a license is cached in the MOSEK environment for reuse. Checking out a license from the license server has a small overhead. Therefore, if a large number of optimizations is performed within a small amount of time, it is efficient to cache the license in the MOSEK environment for later use. This way a number of license check outs from the license server is avoided.

If a license has not been used in the given amount of time, MOSEK will automatically check in the license. To disable license caching set the value to 0.

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 micro seconds between each check of whether a license as 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 $+\infty$.

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 +inf.

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 +inf.

Default value:

2500

- `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 +inf.

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 +inf.

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:

250

- `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 +inf.

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

- `maxnumanz_double_trh`

Corresponding constant:

`mosek.iparam.maxnumanz_double_trh`

Description:

Whenever MOSEK runs out of storage for the A matrix, it will double the value for `maxnumanz` until `maxnumnza` reaches the value of this parameter. When this threshold is reached it will use a slower increase.

Possible Values:

Any number between -inf and +inf.

Default value:

-1

- `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 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 MIP solution exists, then MOSEK generates an initial solution to the MIP by fixing all integer values and solving for the continuous variables.

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 MIP 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_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:**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 criterias 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 criterias 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 criterias 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.conic` Another cone optimizer.

`mosek.optimizertype.nonconvex` The optimizer for nonconvex nonlinear problems.

`mosek.optimizertype.qcone` The Qcone optimizer is used.

`mosek.optimizertype.primal_simplex` The primal simplex optimizer is used.

`mosek.optimizertype.free_simplex` Either the primal or the dual simplex optimizer 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_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.conic` Another cone optimizer.
`mosek.optimizertype.nonconvex` The optimizer for nonconvex nonlinear problems.
`mosek.optimizertype.qcone` The Qcone optimizer is used.
`mosek.optimizertype.primal_simplex` The primal simplex optimizer is used.
`mosek.optimizertype.free_simplex` Either the primal or the dual simplex optimizer 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:

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.conic` Another cone optimizer.
`mosek.optimizertype.nonconvex` The optimizer for nonconvex nonlinear problems.
`mosek.optimizertype.qcone` The Qcone optimizer is used.
`mosek.optimizertype.primal_simplex` The primal simplex optimizer is used.
`mosek.optimizertype.free_simplex` Either the primal or the dual simplex optimizer 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_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 -inf and +inf.

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 +inf.

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`

- `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 +inf.

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 $+\infty$.

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 $+\infty$.

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 $+\infty$.

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 +inf.

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 +inf.

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.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.mpsformattype.strict` It is assumed that the input file satisfies the MPS format strictly.

`mosek.mpsformattype.relaxed` It is assumed that the input file satisfies a slightly relaxed version of the MPS format.

`mosek.mpsformattype.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.mpsformattype.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 the relaxation of the MIP will be 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 chars 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 $+\infty$.

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.conic` Another cone optimizer.

`mosek.optimizertype.nonconvex` The optimizer for nonconvex nonlinear problems.

`mosek.optimizertype.qcone` The Qcone optimizer is used.

`mosek.optimizertype.primal_simplex` The primal simplex optimizer is used.

`mosek.optimizertype.free_simplex` Either the primal or the dual simplex optimizer 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_degen`

Corresponding constant:

`mosek.iparam.sim_degen`

Description:

Controls how aggressive degeneration is approached.

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 $+\text{inf}$.

Default value:

90

- `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_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_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 setbacks are allowed within a simplex optimizer. A setback 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_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 +inf.

Default value:

0

- `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 the problem is scaled 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_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`

- `warning_level`

Corresponding constant:

`mosek.iparam.warning_level`

Description:

Warning level.

Possible Values:

Any number between 0 and $+\infty$.

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 which format the data file has when a task is written to that file using `Task.writedata`.

Possible Values:

`mosek.dataformat.xml` The data file is an XML 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`

14.4 String parameter types

- `mosek.sparam.bas_sol_file_name` 335
Name of the bas solution file.
- `mosek.sparam.data_file_name` 335
Data are read and written to this file.
- `mosek.sparam.debug_file_name` 336
MOSEK debug file.
- `mosek.sparam.feasrepair_name_prefix` 336
Feasibility repair name prefix.
- `mosek.sparam.feasrepair_name_separator` 336
Feasibility repair name separator.
- `mosek.sparam.feasrepair_name_wsumviol` 336
Feasibility repair name violation name.
- `mosek.sparam.int_sol_file_name` 337
Name of the int solution file.
- `mosek.sparam.itr_sol_file_name` 337
Name of the itr solution file.
- `mosek.sparam.param_comment_sign` 337
Solution file comment character.
- `mosek.sparam.param_read_file_name` 337
Modifications to the parameter database is read from this file.
- `mosek.sparam.param_write_file_name` 338
The parameter database is written to this file.
- `mosek.sparam.read_mps_bou_name` 338
Name of the BOUNDS vector used. An empty name means that the first BOUNDS vector is used.
- `mosek.sparam.read_mps_obj_name` 338
Objective name in the MPS file.
- `mosek.sparam.read_mps_ran_name` 338
Name of the RANGE vector used. An empty name means that the first RANGE vector is used.
- `mosek.sparam.read_mps_rhs_name` 339
Name of the RHS used. An empty name means that the first RHS vector is used.
- `mosek.sparam.sensitivity_file_name` 339
Sensitivity report file name.

- `mosek.sparam.sensitivity_res_file_name` 339
Name of the sensitivity report output file.
- `mosek.sparam.sol_filter_xc_low` 339
Solution file filter.
- `mosek.sparam.sol_filter_xc_upr` 340
Solution file filter.
- `mosek.sparam.sol_filter_xx_low` 340
Solution file filter.
- `mosek.sparam.sol_filter_xx_upr` 340
Solution file filter.
- `mosek.sparam.stat_file_name` 341
Statistics file name.
- `mosek.sparam.stat_key` 341
Key used when writing the summary file.
- `mosek.sparam.stat_name` 341
Name used when writing the statistics file.
- `mosek.sparam.write_lp_gen_var_name` 341
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 15

Response codes

(0)	<code>mosek.rescode.ok</code>	412
	No error occurred.	
(50)	<code>mosek.rescode.wrn_open_param_file</code>	420
	The parameter file could not be opened.	
(51)	<code>mosek.rescode.wrn_large_bound</code>	416
	A very large bound in absolute value has been specified.	
(52)	<code>mosek.rescode.wrn_large_lo_bound</code>	417
	A large but finite lower bound in absolute value has been specified.	
(53)	<code>mosek.rescode.wrn_large_up_bound</code>	417
	A large but finite upper bound in absolute value has been specified.	
(57)	<code>mosek.rescode.wrn_large_cj</code>	416
	A numerically large value is specified for one c_j .	
(62)	<code>mosek.rescode.wrn_large_aij</code>	416
	A numerically large value is specified for one $a_{i,j}$.	
(63)	<code>mosek.rescode.wrn_zero_aij</code>	421
	One or more zero elements are specified in A.	
(65)	<code>mosek.rescode.wrn_name_max_len</code>	419
	A name is longer than the buffer that is supposed to hold it.	
(66)	<code>mosek.rescode.wrn_spar_max_len</code>	420
	A value for a string parameter is longer than the buffer that is supposed to hold it.	
(70)	<code>mosek.rescode.wrn_mps_split_rhs_vector</code>	419
	An RHS vector is split into several nonadjacent parts in an MPS file.	
(71)	<code>mosek.rescode.wrn_mps_split_ran_vector</code>	418
	A RANGE vector is split into several nonadjacent parts in an MPS file.	

(72)	<code>mosek.rescode.wrn_mps_split_bou_vector</code>	418
	A BOUNDS vector is split into several nonadjacent parts in an MPS file.	
(80)	<code>mosek.rescode.wrn_lp_old_quad_format</code>	418
	Missing <code>'/2'</code> after quadratic expressions in bound or objective.	
(85)	<code>mosek.rescode.wrn_lp_drop_variable</code>	417
	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)	<code>mosek.rescode.wrn_nz_in_upr_tri</code>	419
	Non-zero elements specified in the upper triangle of a matrix were ignored.	
(201)	<code>mosek.rescode.wrn_dropped_nz_qobj</code>	415
	One or more non-zero elements were dropped in the Q matrix in the objective.	
(250)	<code>mosek.rescode.wrn_ignore_integer</code>	416
	Ignored integer constraints.	
(251)	<code>mosek.rescode.wrn_no_global_optimizer</code>	419
	No global optimizer is available.	
(270)	<code>mosek.rescode.wrn_mio_infeasible_final</code>	418
	The final mixed integer problem with all the integer variables fixed at their optimal values is infeasible.	
(280)	<code>mosek.rescode.wrn_fixed_bound_values</code>	415
	A fixed constraint/variable has been specified using the bound keys but the numerical bounds are different. The variable is fixed at the lower bound.	
(300)	<code>mosek.rescode.wrn_sol_filter</code>	420
	Invalid solution filter is specified.	
(350)	<code>mosek.rescode.wrn_undef_sol_file_name</code>	421
	Undefined name occurred in a solution.	
(400)	<code>mosek.rescode.wrn_too_few_basis_vars</code>	420
	An incomplete basis has been specified. Too few basis variables are specified.	
(405)	<code>mosek.rescode.wrn_too_many_basis_vars</code>	421
	A basis with too many variables has been specified.	
(500)	<code>mosek.rescode.wrn_license_expire</code>	417
	The license expires.	
(501)	<code>mosek.rescode.wrn_license_server</code>	417
	The license server is not responding.	
(502)	<code>mosek.rescode.wrn_empty_name</code>	415
	A variable or constraint name is empty. The output file may be invalid.	

(503)	<code>mosek.rescode.wrn_using_generic_names</code>	421
	The file writer reverts to generic names because a name is blank.	
(505)	<code>mosek.rescode.wrn_license_feature_expire</code>	417
	The license expires.	
(700)	<code>mosek.rescode.wrn_zeros_in_sparse_data</code>	422
	One or more almost zero elements are specified in sparse input data.	
(800)	<code>mosek.rescode.wrn_noncomplete_linear_dependency_check</code>	419
	The linear dependency check(s) was not completed and therefore the A matrix may contain linear dependencies.	
(801)	<code>mosek.rescode.wrn_eliminator_space</code>	415
	The eliminator is skipped at least once due to lack of space.	
(802)	<code>mosek.rescode.wrn_presolve_outofspace</code>	420
	The presolve is incomplete due to lack of space.	
(803)	<code>mosek.rescode.wrn_presolve_bad_precision</code>	420
	The presolve estimates that the model is specified with insufficient precision.	
(804)	<code>mosek.rescode.wrn_write_discarded_cfix</code>	421
	The fixed objective term could not be converted to a variable and was discarded in the output file.	
(1000)	<code>mosek.rescode.err_license</code>	381
	Invalid license.	
(1001)	<code>mosek.rescode.err_license_expired</code>	381
	The license has expired.	
(1002)	<code>mosek.rescode.err_license_version</code>	383
	The license is valid for another version of MOSEK.	
(1005)	<code>mosek.rescode.err_size_license</code>	405
	The problem is bigger than the license.	
(1006)	<code>mosek.rescode.err_prob_license</code>	401
	The software is not licensed to solve the problem.	
(1007)	<code>mosek.rescode.err_file_license</code>	367
	Invalid license file.	
(1008)	<code>mosek.rescode.err_missing_license_file</code>	387
	MOSEK cannot find the license file or license server. Usually this happens if the operating system variable <code>MOSEKLM.LICENSE.FILE</code> is not set up appropriately. Please see the MOSEK installation manual for details.	
(1010)	<code>mosek.rescode.err_size_license_con</code>	405
	The problem has too many constraints to be solved with the available license.	

(1011)	<code>mosek.rescode.err_size_license_var</code>	405
	The problem has too many variables to be solved with the available license.	
(1012)	<code>mosek.rescode.err_size_license_intvar</code>	405
	The problem contains too many integer variables to be solved with the available license.	
(1013)	<code>mosek.rescode.err_optimizer_license</code>	399
	The optimizer required is not licensed.	
(1014)	<code>mosek.rescode.err_flexlm</code>	368
	The FLEXlm license manager reported an error.	
(1015)	<code>mosek.rescode.err_license_server</code>	382
	The license server is not responding.	
(1016)	<code>mosek.rescode.err_license_max</code>	382
	Maximum number of licenses is reached.	
(1017)	<code>mosek.rescode.err_license_moseklm_daemon</code>	382
	The MOSEKLM license manager daemon is not up and running.	
(1018)	<code>mosek.rescode.err_license_feature</code>	381
	A requested feature is not available in the license file(s). Most likely due to an incorrect license system setup.	
(1019)	<code>mosek.rescode.err_platform_not_licensed</code>	401
	A requested license feature is not available for the required platform.	
(1020)	<code>mosek.rescode.err_license_cannot_allocate</code>	381
	The license system cannot allocate the memory required.	
(1021)	<code>mosek.rescode.err_license_cannot_connect</code>	381
	MOSEK cannot connect to the license server. Most likely the license server is not up and running.	
(1025)	<code>mosek.rescode.err_license_invalid_hostid</code>	382
	The host ID specified in the license file does not match the host ID of the computer.	
(1026)	<code>mosek.rescode.err_license_server_version</code>	382
	The version specified in the checkout request is greater than the highest version number the daemon supports.	
(1030)	<code>mosek.rescode.err_open_dl</code>	398
	A dynamic link library could not be opened.	
(1035)	<code>mosek.rescode.err_older_dll</code>	398
	The dynamic link library is older than the specified version.	
(1036)	<code>mosek.rescode.err_newer_dll</code>	394
	The dynamic link library is newer than the specified version.	
(1040)	<code>mosek.rescode.err_link_file_dll</code>	383
	A file cannot be linked to a stream in the DLL version.	

(1045)	<code>mosek.rescode.err_thread_mutex_init</code>	407
	Could not initialize a mutex.	
(1046)	<code>mosek.rescode.err_thread_mutex_lock</code>	407
	Could not lock a mutex.	
(1047)	<code>mosek.rescode.err_thread_mutex_unlock</code>	408
	Could not unlock a mutex.	
(1048)	<code>mosek.rescode.err_thread_create</code>	407
	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)	<code>mosek.rescode.err_thread_cond_init</code>	407
	Could not initialize a condition.	
(1050)	<code>mosek.rescode.err_unknown</code>	409
	Unknown error.	
(1051)	<code>mosek.rescode.err_space</code>	406
	Out of space.	
(1052)	<code>mosek.rescode.err_file_open</code>	367
	Error while opening a file.	
(1053)	<code>mosek.rescode.err_file_read</code>	367
	File read error.	
(1054)	<code>mosek.rescode.err_file_write</code>	368
	File write error.	
(1055)	<code>mosek.rescode.err_data_file_ext</code>	366
	The data file format cannot be determined from the file name.	
(1056)	<code>mosek.rescode.err_invalid_file_name</code>	378
	An invalid file name has been specified.	
(1057)	<code>mosek.rescode.err_invalid_sol_file_name</code>	379
	An invalid file name has been specified.	
(1058)	<code>mosek.rescode.err_invalid_mbt_file</code>	378
	A MOSEK binary task file is invalid.	
(1059)	<code>mosek.rescode.err_end_of_file</code>	366
	End of file reached.	
(1060)	<code>mosek.rescode.err_null_env</code>	396
	<code>env</code> is a NULL pointer.	
(1061)	<code>mosek.rescode.err_null_task</code>	397
	<code>task</code> is a NULL pointer.	

(1062)	<code>mosek.rescode.err_invalid_stream</code>	379
	An invalid stream is referenced.	
(1063)	<code>mosek.rescode.err_no_init_env</code>	395
	<code>env</code> is not initialized.	
(1064)	<code>mosek.rescode.err_invalid_task</code>	379
	The <code>task</code> is invalid.	
(1065)	<code>mosek.rescode.err_null_pointer</code>	396
	An argument to a function is unexpectedly a NULL pointer.	
(1070)	<code>mosek.rescode.err_null_name</code>	396
	An all blank name has been specified.	
(1071)	<code>mosek.rescode.err_dup_name</code>	366
	An error occurred while reading an MPS file..	
(1075)	<code>mosek.rescode.err_invalid_obj_name</code>	379
	An invalid objective name is specified.	
(1080)	<code>mosek.rescode.err_space_leaking</code>	406
	MOSEK is leaking memory. This can be due to either an incorrect use of MOSEK or a bug.	
(1081)	<code>mosek.rescode.err_space_no_info</code>	407
	No available information about the space usage.	
(1090)	<code>mosek.rescode.err_read_format</code>	402
	The specified format cannot be read.	
(1100)	<code>mosek.rescode.err_mps_file</code>	388
	An error occurred while reading an MPS file.	
(1101)	<code>mosek.rescode.err_mps_inv_field</code>	388
	A field in the MPS file is invalid. Probably it is too wide.	
(1102)	<code>mosek.rescode.err_mps_inv_marker</code>	388
	An invalid marker has been specified in the MPS file.	
(1103)	<code>mosek.rescode.err_mps_null_con_name</code>	390
	An empty constraint name is used in an MPS file.	
(1104)	<code>mosek.rescode.err_mps_null_var_name</code>	390
	An empty variable name is used in an MPS file.	
(1105)	<code>mosek.rescode.err_mps_undef_con_name</code>	391
	An undefined constraint name occurred in an MPS file.	
(1106)	<code>mosek.rescode.err_mps_undef_var_name</code>	392
	An undefined variable name occurred in an MPS file.	
(1107)	<code>mosek.rescode.err_mps_inv_con_key</code>	388
	An invalid constraint key occurred in an MPS file.	

(1108)	<code>mosek.rescode.err_mps_inv_bound_key</code>	388
	An invalid bound key occurred in an MPS file.	
(1109)	<code>mosek.rescode.err_mps_inv_sec_name</code>	389
	An invalid section name occurred in an MPS file.	
(1110)	<code>mosek.rescode.err_mps_no_objective</code>	390
	No objective is defined in an MPS file.	
(1111)	<code>mosek.rescode.err_mps_splitted_var</code>	391
	A variable is split in an MPS data file.	
(1112)	<code>mosek.rescode.err_mps_mul_con_name</code>	389
	A constraint name was specified multiple times in the ROWS section.	
(1113)	<code>mosek.rescode.err_mps_mul_qsec</code>	390
	Multiple QSECTIONs are specified for a constraint in the MPS data file.	
(1114)	<code>mosek.rescode.err_mps_mul_qobj</code>	390
	The Q term in the objective is specified multiple times in the MPS data file.	
(1115)	<code>mosek.rescode.err_mps_inv_sec_order</code>	389
	The sections in the MPS data file are not in the correct order.	
(1116)	<code>mosek.rescode.err_mps_mul_csec</code>	390
	Multiple CSECTIONs are given the same name.	
(1117)	<code>mosek.rescode.err_mps_cone_type</code>	387
	Invalid cone type specified in a CSECTION.	
(1118)	<code>mosek.rescode.err_mps_cone_overlap</code>	387
	A variable is specified to be a member of several cones.	
(1119)	<code>mosek.rescode.err_mps_cone_repeat</code>	387
	A variable is repeated within the CSECTION.	
(1122)	<code>mosek.rescode.err_mps_invalid_objsense</code>	389
	An invalid objective sense is specified.	
(1125)	<code>mosek.rescode.err_mps_tab_in_field2</code>	391
	A tab char occurred in field 2.	
(1126)	<code>mosek.rescode.err_mps_tab_in_field3</code>	391
	A tab char occurred in field 3.	
(1127)	<code>mosek.rescode.err_mps_tab_in_field5</code>	391
	A tab char occurred in field 5.	
(1128)	<code>mosek.rescode.err_mps_invalid_obj_name</code>	389
	An invalid objective name is specified.	
(1130)	<code>mosek.rescode.err_ord_invalid_branch_dir</code>	399
	An invalid branch direction key is specified.	

(1131)	<code>mosek.rescode.err_ord_invalid</code>	399
	Invalid content in branch ordering file.	
(1150)	<code>mosek.rescode.err_lp_incompatible</code>	384
	The problem cannot be written to an LP formatted file.	
(1151)	<code>mosek.rescode.err_lp_empty</code>	383
	The problem cannot be written to an LP formatted file.	
(1152)	<code>mosek.rescode.err_lp_dup_slack_name</code>	383
	The name of the slack variable added to a ranged constraint already exists.	
(1153)	<code>mosek.rescode.err_write_mps_invalid_name</code>	411
	An invalid name is created while writing an MPS file. Usually this will make the MPS file unreadable.	
(1154)	<code>mosek.rescode.err_lp_invalid_var_name</code>	384
	A variable name is invalid when used in an LP formatted file.	
(1155)	<code>mosek.rescode.err_lp_free_constraint</code>	384
	Free constraints cannot be written in LP file format.	
(1156)	<code>mosek.rescode.err_write_opf_invalid_var_name</code>	411
	Empty variable names cannot be written to OPF files.	
(1157)	<code>mosek.rescode.err_lp_file_format</code>	383
	Syntax error in an LP file.	
(1158)	<code>mosek.rescode.err_write_lp_format</code>	410
	Problem cannot be written as an LP file.	
(1160)	<code>mosek.rescode.err_lp_format</code>	383
	Syntax error in an LP file.	
(1161)	<code>mosek.rescode.err_write_lp_non_unique_name</code>	411
	An auto-generated name is not unique.	
(1162)	<code>mosek.rescode.err_read_lp_nonexisting_name</code>	403
	A variable never occurred in objective or constraints.	
(1163)	<code>mosek.rescode.err_lp_write_conic_problem</code>	384
	The problem contains cones that cannot be written to an LP formatted file.	
(1164)	<code>mosek.rescode.err_lp_write_geco_problem</code>	384
	The problem contains general convex terms that cannot be written to an LP formatted file.	
(1165)	<code>mosek.rescode.err_name_max_len</code>	392
	A name is longer than the buffer that is supposed to hold it.	
(1168)	<code>mosek.rescode.err_opf_format</code>	398
	Syntax error in an OPF file	

(1170)	<code>mosek.rescode.err_invalid_name_in_sol_file</code>	378
	An invalid name occurred in a solution file.	
(1197)	<code>mosek.rescode.err_argument_lenneq</code>	362
	Incorrect length of arguments.	
(1198)	<code>mosek.rescode.err_argument_type</code>	363
	Incorrect argument type.	
(1199)	<code>mosek.rescode.err_nr_arguments</code>	396
	Incorrect number of function arguments.	
(1200)	<code>mosek.rescode.err_in_argument</code>	369
	A function argument is incorrect.	
(1201)	<code>mosek.rescode.err_argument_dimension</code>	362
	A function argument is of incorrect dimension.	
(1203)	<code>mosek.rescode.err_index_is_too_small</code>	370
	An index in an argument is too small.	
(1204)	<code>mosek.rescode.err_index_is_too_large</code>	370
	An index in an argument is too large.	
(1205)	<code>mosek.rescode.err_param_name</code>	400
	The parameter name is not correct.	
(1206)	<code>mosek.rescode.err_param_name_dou</code>	400
	The parameter name is not correct for a double parameter.	
(1207)	<code>mosek.rescode.err_param_name_int</code>	400
	The parameter name is not correct for an integer parameter.	
(1208)	<code>mosek.rescode.err_param_name_str</code>	400
	The parameter name is not correct for a string parameter.	
(1210)	<code>mosek.rescode.err_param_index</code>	399
	Parameter index is out of range.	
(1215)	<code>mosek.rescode.err_param_is_too_large</code>	399
	The parameter value is too large.	
(1216)	<code>mosek.rescode.err_param_is_too_small</code>	399
	The parameter value is too small.	
(1217)	<code>mosek.rescode.err_param_value_str</code>	401
	The parameter value string is incorrect.	
(1218)	<code>mosek.rescode.err_param_type</code>	400
	The parameter type is invalid.	
(1219)	<code>mosek.rescode.err_inf_dou_index</code>	370
	A double information index is out of range for the specified type.	

(1220)	<code>mosek.rescode.err_inf_int_index</code>	371
	An integer information index is out of range for the specified type.	
(1221)	<code>mosek.rescode.err_index_arr_is_too_small</code>	370
	An index in an array argument is too small.	
(1222)	<code>mosek.rescode.err_index_arr_is_too_large</code>	369
	An index in an array argument is too large.	
(1230)	<code>mosek.rescode.err_inf_dou_name</code>	370
	A double information name is invalid.	
(1231)	<code>mosek.rescode.err_inf_int_name</code>	371
	A integer information name is invalid.	
(1232)	<code>mosek.rescode.err_inf_type</code>	371
	The information type is invalid.	
(1235)	<code>mosek.rescode.err_index</code>	369
	An index is out of range.	
(1236)	<code>mosek.rescode.err_whichsol</code>	410
	The solution defined by <code>compwhichsol</code> does not exists.	
(1237)	<code>mosek.rescode.err_solitem</code>	406
	The solution item number <code>solitem</code> is invalid. Please note <code>mosek.solitem.snx</code> is invalid for the basic solution.	
(1238)	<code>mosek.rescode.err_whichitem_not_allowed</code>	410
	<code>whichitem</code> is unacceptable.	
(1240)	<code>mosek.rescode.err_maxnumcon</code>	385
	The maximum number of constraints specified is smaller than the number of constraints in the task.	
(1241)	<code>mosek.rescode.err_maxnumvar</code>	386
	The maximum number of variables specified is smaller than the number of variables in the task.	
(1242)	<code>mosek.rescode.err_maxnumanz</code>	385
	The maximum number of non-zeros specified for A is smaller than the number of non-zeros in the current A .	
(1243)	<code>mosek.rescode.err_maxnumqnz</code>	386
	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)	<code>mosek.rescode.err_numconlim</code>	397
	Maximum number of constraints limit is exceeded.	
(1251)	<code>mosek.rescode.err_numvarlim</code>	397
	Maximum number of variables limit is exceeded.	

(1252)	<code>mosek.rescode.err_too_small_maxnumanz</code>	408
	Maximum number of non-zeros allowed in A is too small.	
(1253)	<code>mosek.rescode.err_inv_aptre</code>	372
	<code>aptre[j]</code> is strictly smaller than <code>aptrb[j]</code> for some j .	
(1254)	<code>mosek.rescode.err_mul_a_element</code>	392
	An element in A is defined multiple times.	
(1255)	<code>mosek.rescode.err_inv_bk</code>	372
	Invalid bound key.	
(1256)	<code>mosek.rescode.err_inv_bkc</code>	372
	Invalid bound key is specified for a constraint.	
(1257)	<code>mosek.rescode.err_inv_bkx</code>	372
	An invalid bound key is specified for a variable.	
(1258)	<code>mosek.rescode.err_inv_var_type</code>	377
	An invalid variable type is specified for a variable.	
(1259)	<code>mosek.rescode.err_solver_probtype</code>	406
	Problem type does not match the chosen optimizer.	
(1260)	<code>mosek.rescode.err_objective_range</code>	398
	Empty objective range.	
(1261)	<code>mosek.rescode.err_first</code>	368
	Invalid <code>first</code> .	
(1262)	<code>mosek.rescode.err_last</code>	380
	Invalid <code>last</code> .	
(1263)	<code>mosek.rescode.err_negative_surplus</code>	394
	Negative surplus.	
(1264)	<code>mosek.rescode.err_negative_append</code>	393
	Cannot append a negative number.	
(1265)	<code>mosek.rescode.err_undef_solution</code>	408
	The required solution is not defined.	
(1266)	<code>mosek.rescode.err_basis</code>	363
	An invalid basis is specified. Either too many or too few basis variables are specified.	
(1267)	<code>mosek.rescode.err_inv_skc</code>	376
	Invalid value in <code>skc</code> .	
(1268)	<code>mosek.rescode.err_inv_skx</code>	376
	Invalid value in <code>skx</code> .	
(1269)	<code>mosek.rescode.err_inv_sk_str</code>	376
	Invalid status key string encountered.	

(1270)	<code>mosek.rescode.err_inv_sk</code>	376
	Invalid status key code.	
(1271)	<code>mosek.rescode.err_inv_cone_type_str</code>	373
	Invalid cone type string encountered.	
(1272)	<code>mosek.rescode.err_inv_cone_type</code>	372
	Invalid cone type code is encountered.	
(1274)	<code>mosek.rescode.err_inv_skn</code>	376
	Invalid value in <code>skn</code> .	
(1280)	<code>mosek.rescode.err_inv_name_item</code>	373
	An invalid name item code is used.	
(1281)	<code>mosek.rescode.err_pro_item</code>	401
	An invalid problem is used.	
(1283)	<code>mosek.rescode.err_invalid_format_type</code>	378
	Invalid format type.	
(1285)	<code>mosek.rescode.err_firsti</code>	368
	Invalid <code>firsti</code> .	
(1286)	<code>mosek.rescode.err_lasti</code>	380
	Invalid <code>lasti</code> .	
(1287)	<code>mosek.rescode.err_firstj</code>	368
	Invalid <code>firstj</code> .	
(1288)	<code>mosek.rescode.err_lastj</code>	380
	Invalid <code>lastj</code> .	
(1290)	<code>mosek.rescode.err_nonlinear_equality</code>	395
	The model contains a nonlinear equality which defines a nonconvex set.	
(1291)	<code>mosek.rescode.err_nonconvex</code>	395
	The optimization problem is nonconvex.	
(1292)	<code>mosek.rescode.err_nonlinear_ranged</code>	396
	The model contains a nonlinear ranged constraint which by definition defines a nonconvex set.	
(1293)	<code>mosek.rescode.err_con_q_not_psd</code>	364
	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)	<code>mosek.rescode.err_con_q_not_nsd</code>	364
	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)	<code>mosek.rescode.err_obj_q_not_psd</code>	397
	The quadratic coefficient matrix in the objective is not positive semi-definite as expected for a minimization problem.	

(1296)	<code>mosek.rescode.err_obj_q_not_nsd</code>	397
	The quadratic coefficient matrix in the objective is not negative semi-definite as expected for a maximization problem.	
(1299)	<code>mosek.rescode.err_argument_perm_array</code>	363
	An invalid permutation array is specified.	
(1300)	<code>mosek.rescode.err_cone_index</code>	365
	An index of a non-existing cone has been specified.	
(1301)	<code>mosek.rescode.err_cone_size</code>	365
	A cone with too few members is specified.	
(1302)	<code>mosek.rescode.err_cone_overlap</code>	365
	A new cone which variables overlap with an existing cone has been specified.	
(1303)	<code>mosek.rescode.err_cone_rep_var</code>	365
	A variable is included multiple times in the cone.	
(1304)	<code>mosek.rescode.err_maxnumcone</code>	385
	The value specified for <code>maxnumcone</code> is too small.	
(1305)	<code>mosek.rescode.err_cone_type</code>	365
	Invalid cone type specified.	
(1306)	<code>mosek.rescode.err_cone_type_str</code>	365
	Invalid cone type specified.	
(1310)	<code>mosek.rescode.err_remove_cone_variable</code>	403
	A variable cannot be removed because it will make a cone invalid.	
(1350)	<code>mosek.rescode.err_sol_file_number</code>	406
	An invalid number is specified in a solution file.	
(1375)	<code>mosek.rescode.err_huge_c</code>	369
	A huge value in absolute size is specified for one c_j .	
(1400)	<code>mosek.rescode.err_infinite_bound</code>	371
	A finite bound value is too large in absolute value.	
(1401)	<code>mosek.rescode.err_inv_qobj_subi</code>	375
	Invalid value in <code>qosubi</code> .	
(1402)	<code>mosek.rescode.err_inv_qobj_subj</code>	375
	Invalid value in <code>qosubj</code> .	
(1403)	<code>mosek.rescode.err_inv_qobj_val</code>	375
	Invalid value in <code>qoval</code> .	
(1404)	<code>mosek.rescode.err_inv_qcon_subk</code>	375
	Invalid value in <code>qconsubk</code> .	

(1405)	<code>mosek.rescode.err_inv_qcon_subi</code>	374
	Invalid value in <code>qcon_subi</code> .	
(1406)	<code>mosek.rescode.err_inv_qcon_subj</code>	374
	Invalid value in <code>qcon_subj</code> .	
(1407)	<code>mosek.rescode.err_inv_qcon_val</code>	375
	Invalid value in <code>qcon_val</code> .	
(1408)	<code>mosek.rescode.err_qcon_subi_too_small</code>	402
	Invalid value in <code>qcon_subi</code> .	
(1409)	<code>mosek.rescode.err_qcon_subi_too_large</code>	402
	Invalid value in <code>qcon_subi</code> .	
(1415)	<code>mosek.rescode.err_qobj_upper_triangle</code>	402
	An element in the upper triangle of Q^o is specified. Only elements in the lower triangle should be specified.	
(1417)	<code>mosek.rescode.err_qcon_upper_triangle</code>	402
	An element in the upper triangle of a Q^k is specified. Only elements in the lower triangle should be specified.	
(1430)	<code>mosek.rescode.err_user_func_ret</code>	409
	An user function reported an error.	
(1431)	<code>mosek.rescode.err_user_func_ret_data</code>	409
	An user function returned invalid data.	
(1432)	<code>mosek.rescode.err_user_nlo_func</code>	410
	The user-defined nonlinear function reported an error.	
(1433)	<code>mosek.rescode.err_user_nlo_eval</code>	409
	The user-defined nonlinear function reported an error.	
(1440)	<code>mosek.rescode.err_user_nlo_eval_hessubi</code>	409
	The user-defined nonlinear function reported an invalid subscript in the Hessian.	
(1441)	<code>mosek.rescode.err_user_nlo_eval_hesssubj</code>	410
	The user-defined nonlinear function reported an invalid subscript in the Hessian.	
(1445)	<code>mosek.rescode.err_invalid_objective_sense</code>	379
	An invalid objective sense is specified.	
(1446)	<code>mosek.rescode.err_undefined_objective_sense</code>	408
	The objective sense has not been specified before the optimization.	
(1449)	<code>mosek.rescode.err_y_is_undefined</code>	411
	The solution item y is undefined.	
(1450)	<code>mosek.rescode.err_nan_in_double_data</code>	393
	An invalid floating point value was used in some double data.	

(1461)	<code>mosek.rescode.err_nan_in_blc</code>	392
	l^c contains an invalid floating point value, i.e. a NaN.	
(1462)	<code>mosek.rescode.err_nan_in_buc</code>	393
	u^c contains an invalid floating point value, i.e. a NaN.	
(1470)	<code>mosek.rescode.err_nan_in_c</code>	393
	c contains an invalid floating point value, i.e. a NaN.	
(1471)	<code>mosek.rescode.err_nan_in_blx</code>	392
	l^x contains an invalid floating point value, i.e. a NaN.	
(1472)	<code>mosek.rescode.err_nan_in_bux</code>	393
	u^x contains an invalid floating point value, i.e. a NaN.	
(1500)	<code>mosek.rescode.err_inv_problem</code>	374
	Invalid problem type. Probably a nonconvex problem has been specified.	
(1501)	<code>mosek.rescode.err_mixed_problem</code>	387
	The problem contains both conic and nonlinear constraints.	
(1550)	<code>mosek.rescode.err_inv_optimizer</code>	374
	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)	<code>mosek.rescode.err_mio_no_optimizer</code>	386
	No optimizer is available for the current class of integer optimization problems.	
(1552)	<code>mosek.rescode.err_no_optimizer_var_type</code>	395
	No optimizer is available for this class of optimization problems.	
(1553)	<code>mosek.rescode.err_mio_not_loaded</code>	386
	The mixed-integer optimizer is not loaded.	
(1580)	<code>mosek.rescode.err_postsolve</code>	401
	An error occurred during the postsolve. Please contact MOSEK support.	
(1600)	<code>mosek.rescode.err_no_basis_sol</code>	394
	No basic solution is defined.	
(1610)	<code>mosek.rescode.err_basis_factor</code>	363
	The factorization of the basis is invalid.	
(1615)	<code>mosek.rescode.err_basis_singular</code>	363
	The basis is singular and hence cannot be factored.	
(1650)	<code>mosek.rescode.err_factor</code>	366
	An error occurred while factorizing a matrix.	
(1700)	<code>mosek.rescode.err_feasrepair_cannot_relax</code>	366
	An optimization problem cannot be relaxed. This is the case e.g. for general nonlinear optimization problems.	

(1701)	<code>mosek.rescode.err_feasrepair_solving_relaxed</code>	367
	The relaxed problem could not be solved to optimality. Please consult the log file for further details.	
(1702)	<code>mosek.rescode.err_feasrepair_inconsistent_bound</code>	367
	The upper bound is less than the lower bound for a variable or a constraint. Please correct this before running the feasibility repair.	
(1800)	<code>mosek.rescode.err_invalid_compression</code>	377
	Invalid compression type.	
(1801)	<code>mosek.rescode.err_invalid_iomode</code>	378
	Invalid io mode.	
(2000)	<code>mosek.rescode.err_no_primal_infeas_cer</code>	395
	A certificate of primal infeasibility is not available.	
(2001)	<code>mosek.rescode.err_no_dual_infeas_cer</code>	394
	A certificate of infeasibility is not available.	
(2500)	<code>mosek.rescode.err_no_solution_in_callback</code>	395
	The required solution is not available.	
(2501)	<code>mosek.rescode.err_inv_marki</code>	373
	Invalid value in marki.	
(2502)	<code>mosek.rescode.err_inv_markj</code>	373
	Invalid value in markj.	
(2503)	<code>mosek.rescode.err_inv_numi</code>	373
	Invalid numi.	
(2504)	<code>mosek.rescode.err_inv_numj</code>	374
	Invalid numj.	
(2505)	<code>mosek.rescode.err_cannot_clone_nl</code>	363
	A task with a nonlinear function call-back cannot be cloned.	
(2506)	<code>mosek.rescode.err_cannot_handle_nl</code>	364
	A function cannot handle a task with nonlinear function call-backs.	
(2520)	<code>mosek.rescode.err_invalid_accmode</code>	377
	An invalid access mode is specified.	
(2550)	<code>mosek.rescode.err_mbt_incompatible</code>	386
	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)	<code>mosek.rescode.err_lu_max_num_tries</code>	385
	Could not compute the LU factors of the matrix within the maximum number of allowed tries.	

(2900)	<code>mosek.rescode.err_invalid_utf8</code>	380
	An invalid UTF8 string is encountered.	
(2901)	<code>mosek.rescode.err_invalid_wchar</code>	380
	An invalid <code>wchar</code> string is encountered.	
(2950)	<code>mosek.rescode.err_no_dual_for_itg_sol</code>	394
	No dual information is available for the integer solution.	
(3000)	<code>mosek.rescode.err_internal</code>	371
	An internal error occurred. Please report this problem.	
(3001)	<code>mosek.rescode.err_api_array_too_small</code>	361
	An input array was too short.	
(3002)	<code>mosek.rescode.err_api_cb_connect</code>	362
(3003)	<code>mosek.rescode.err_api_nl_data</code>	362
(3004)	<code>mosek.rescode.err_api_callback</code>	361
(3005)	<code>mosek.rescode.err_api_fatal_error</code>	362
	An internal error occurred in the API. Please report this problem.	
(3050)	<code>mosek.rescode.err_sen_format</code>	403
	Syntax error in sensitivity analysis file.	
(3051)	<code>mosek.rescode.err_sen_undef_name</code>	405
	An undefined name was encountered in the sensitivity analysis file.	
(3052)	<code>mosek.rescode.err_sen_index_range</code>	404
	Index out of range in the sensitivity analysis file.	
(3053)	<code>mosek.rescode.err_sen_bound_invalid_up</code>	403
	Analysis of upper bound requested for an index, where no upper bound exists.	
(3054)	<code>mosek.rescode.err_sen_bound_invalid_lo</code>	403
	Analysis of lower bound requested for an index, where no lower bound exists.	
(3055)	<code>mosek.rescode.err_sen_index_invalid</code>	404
	Invalid range given in the sensitivity file.	
(3056)	<code>mosek.rescode.err_sen_invalid_regexp</code>	404
	Syntax error in regexp or regexp longer than 1024.	
(3057)	<code>mosek.rescode.err_sen_solution_status</code>	404
	No optimal solution found to the original problem given for sensitivity analysis.	
(3058)	<code>mosek.rescode.err_sen_numerical</code>	404
	Numerical difficulties encountered performing the sensitivity analysis.	

(3059)	<code>mosek.rescode.err_concurrent_optimizer</code>	364
	An unsupported optimizer was chosen for use with the concurrent optimizer.	
(3100)	<code>mosek.rescode.err_unb_step_size</code>	408
	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)	<code>mosek.rescode.err_identical_tasks</code>	369
	Some tasks related to this function call were identical. Unique tasks were expected.	
(3200)	<code>mosek.rescode.err_invalid_branch_direction</code>	377
	An invalid branching direction is specified.	
(3201)	<code>mosek.rescode.err_invalid_branch_priority</code>	377
	An invalid branching priority is specified. It should be nonnegative.	
(3500)	<code>mosek.rescode.err_internal_test_failed</code>	371
	An internal unit test function failed.	
(3600)	<code>mosek.rescode.err_xml_invalid_problem_type</code>	411
	The problem type is not supported by the XML format.	
(3700)	<code>mosek.rescode.err_invalid_ampl_stub</code>	377
	Invalid AMPL stub.	
(3999)	<code>mosek.rescode.err_api_internal</code>	362
(4000)	<code>mosek.rescode.trm_max_iterations</code>	412
	The optimizer terminated at the maximum number of iterations.	
(4001)	<code>mosek.rescode.trm_max_time</code>	413
	The optimizer terminated at the maximum amount of time.	
(4002)	<code>mosek.rescode.trm_objective_range</code>	414
	The optimizer terminated on the bound of the objective range.	
(4003)	<code>mosek.rescode.trm_mio_near_rel_gap</code>	413
	The mixed-integer optimizer terminated because the near optimal relative gap tolerance was satisfied.	
(4004)	<code>mosek.rescode.trm_mio_near_abs_gap</code>	413
	The mixed-integer optimizer terminated because the near optimal absolute gap tolerance was satisfied.	
(4005)	<code>mosek.rescode.trm_user_break</code>	415
	The optimizer terminated due to a user break.	

- (4006) `mosek.rescode.trm_stall` 414
 The optimizer terminated due to slow progress. Normally there are three possible reasons for this: Either a bug in MOSEK, the problem is badly formulated, or, in case of nonlinear problems, the nonlinear call-back functions are incorrect.
 Please contact MOSEK support if this happens.
- (4007) `mosek.rescode.trm_user_callback` 415
 The optimizer terminated due to the return of the user-defined call-back function.
- (4008) `mosek.rescode.trm_mio_num_relaxs` 413
 The mixed-integer optimizer terminated as the maximum number of relaxations was reached.
- (4009) `mosek.rescode.trm_mio_num_branches` 413
 The mixed-integer optimizer terminated as to the maximum number of branches was reached.
- (4015) `mosek.rescode.trm_num_max_num_int_solutions` 414
 The mixed-integer optimizer terminated as the maximum number of feasible solutions was reached.
- (4020) `mosek.rescode.trm_max_num_setbacks` 412
 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` 414
 The optimizer terminated due to numerical problems.
- (4030) `mosek.rescode.trm_internal` 412
 The optimizer terminated due to some internal reason. Please contact MOSEK support.
- (4031) `mosek.rescode.trm_internal_stop` 412
 The optimizer terminated for internal reasons. Please contact MOSEK support.

- `err_api_array_too_small`

Corresponding constant:

`mosek.rescode.err_api_array_too_small`

Description:

An input array was too short.

Response message string:

"The input array '0' is too short in call to '1'."

- `err_api_callback`

Corresponding constant:

`mosek.rescode.err_api_callback`

Response message string:

" "

- `err_api_cb_connect`

Corresponding constant:

`mosek.rescode.err_api_cb_connect`

Response message string:

“”

- `err_api_fatal_error`

Corresponding constant:

`mosek.rescode.err_api_fatal_error`

Description:

An internal error occurred in the API. Please report this problem.

Response message string:

“An internal error occurred in the %s API: %s”

- `err_api_internal`

Corresponding constant:

`mosek.rescode.err_api_internal`

Response message string:

“An internal fatal error occurred in an interface function”

- `err_api_nl_data`

Corresponding constant:

`mosek.rescode.err_api_nl_data`

Response message string:

“”

- `err_argument_dimension`

Corresponding constant:

`mosek.rescode.err_argument_dimension`

Description:

A function argument is of incorrect dimension.

Response message string:

“The argument '%s' is of incorrect dimension.”

- `err_argument_lenneq`

Corresponding constant:

`mosek.rescode.err_argument_lenneq`

Description:

Incorrect length of arguments.

Response message string:

“Incorrect argument length. The arguments %s are expected to be of equal length. The length of the arguments was %s.”

- `err_argument_perm_array`

Corresponding constant:

`mosek.rescode.err_argument_perm_array`

Description:

An invalid permutation array is specified.

Response message string:

“An invalid permutation array named '%s' is supplied. Position %d has the invalid value %d.”

- `err_argument_type`

Corresponding constant:

`mosek.rescode.err_argument_type`

Description:

Incorrect argument type.

Response message string:

“Incorrect type in %s argument number: '%d'.”

- `err_basis`

Corresponding constant:

`mosek.rescode.err_basis`

Description:

An invalid basis is specified. Either too many or too few basis variables are specified.

Response message string:

“%d number of basis variables are specified. %d are expected.”

- `err_basis_factor`

Corresponding constant:

`mosek.rescode.err_basis_factor`

Description:

The factorization of the basis is invalid.

Response message string:

“The factorization of the basis is invalid.”

- `err_basis_singular`

Corresponding constant:

`mosek.rescode.err_basis_singular`

Description:

The basis is singular and hence cannot be factored.

Response message string:

“The basis is singular.”

- `err_cannot_clone_nl`

Corresponding constant:

`mosek.rescode.err_cannot_clone_nl`

Description:

A task with a nonlinear function call-back cannot be cloned.

Response message string:

“A task with a nonlinear function call-back cannot be cloned.”

- `err_cannot_handle_nl`

Corresponding constant:

`mosek.rescode.err_cannot_handle_nl`

Description:

A function cannot handle a task with nonlinear function call-backs.

Response message string:

“A function cannot handle a task with nonlinear function call-backs.”

- `err_con_q_not_nsd`

Corresponding constant:

`mosek.rescode.err_con_q_not_nsd`

Description:

The quadratic constraint matrix is not negative semi-definite as expected for a constraint with finite lower bound. This results in a nonconvex problem.

Response message string:

“The quadratic constraint matrix in constraint '%s'(%d) is not negative semi-definite as expected for a constraint with finite lower bound.”

- `err_con_q_not_psd`

Corresponding constant:

`mosek.rescode.err_con_q_not_psd`

Description:

The quadratic constraint matrix is not positive semi-definite as expected for a constraint with finite upper bound. This results in a nonconvex problem.

Response message string:

“The quadratic constraint matrix in constraint '%s'(%d) is not positive semi-definite as expected for a constraint with finite upper bound.”

- `err_concurrent_optimizer`

Corresponding constant:

`mosek.rescode.err_concurrent_optimizer`

Description:

An unsupported optimizer was chosen for use with the concurrent optimizer.

Response message string:

“Optimizer '%s' is unsupported in the concurrent optimizer.”

- `err_cone_index`

Corresponding constant:

`mosek.rescode.err_cone_index`

Description:

An index of a non-existing cone has been specified.

Response message string:

“No cone has index '%d'.”

- `err_cone_overlap`

Corresponding constant:

`mosek.rescode.err_cone_overlap`

Description:

A new cone which variables overlap with an existing cone has been specified.

Response message string:

“Variable '%s' (%d) is a member of cone '%s' (%d) and cone '%s' (%d).”

- `err_cone_rep_var`

Corresponding constant:

`mosek.rescode.err_cone_rep_var`

Description:

A variable is included multiple times in the cone.

Response message string:

“Variable '%s' (%d) are included multiple times in cone '%s' (%d).”

- `err_cone_size`

Corresponding constant:

`mosek.rescode.err_cone_size`

Description:

A cone with too few members is specified.

Response message string:

“A cone with too few member is specified. At least %d members are required for cones of type %s.”

- `err_cone_type`

Corresponding constant:

`mosek.rescode.err_cone_type`

Description:

Invalid cone type specified.

Response message string:

“%d is an invalid cone type specified.”

- `err_cone_type_str`

Corresponding constant:

`mosek.rescode.err_cone_type_str`

Description:

Invalid cone type specified.

Response message string:

“%d is an invalid cone type specified.”

- `err_data_file_ext`

Corresponding constant:

`mosek.rescode.err_data_file_ext`

Description:

The data file format cannot be determined from the file name.

Response message string:

“The data file format cannot be determined from the file name '%s'”

- `err_dup_name`

Corresponding constant:

`mosek.rescode.err_dup_name`

Description:

An error occurred while reading an MPS file..

Response message string:

“Name '%s' is already assigned for an item %s.”

- `err_end_of_file`

Corresponding constant:

`mosek.rescode.err_end_of_file`

Description:

End of file reached.

Response message string:

“End of file reached.”

- `err_factor`

Corresponding constant:

`mosek.rescode.err_factor`

Description:

An error occurred while factorizing a matrix.

Response message string:

“An error occurred while factorizing a matrix.”

- `err_feasrepair_cannot_relax`

Corresponding constant:

`mosek.rescode.err_feasrepair_cannot_relax`

Description:

An optimization problem cannot be relaxed. This is the case e.g. for general nonlinear optimization problems.

Response message string:

“An optimization problem cannot be relaxed.”

- `err_feasrepair_inconsistent_bound`

Corresponding constant:

`mosek.rescode.err_feasrepair_inconsistent_bound`

Description:

The upper bound is less than the lower bound for a variable or a constraint. Please correct this before running the feasibility repair.

Response message string:

“The %s '%s' with index '%d' has lower bound larger than upper bound.”

- `err_feasrepair_solving_relaxed`

Corresponding constant:

`mosek.rescode.err_feasrepair_solving_relaxed`

Description:

The relaxed problem could not be solved to optimality. Please consult the log file for further details.

Response message string:

“The relaxed problem could not be solved to optimality.”

- `err_file_license`

Corresponding constant:

`mosek.rescode.err_file_license`

Description:

Invalid license file.

Response message string:

“Invalid license file.”

- `err_file_open`

Corresponding constant:

`mosek.rescode.err_file_open`

Description:

Error while opening a file.

Response message string:

“An error occurred while opening file '%s'.”

- `err_file_read`

Corresponding constant:

`mosek.rescode.err_file_read`

Description:

File read error.

Response message string:

"An error occurred while reading file '%s'."

- `err_file_write`

Corresponding constant:

`mosek.rescode.err_file_write`

Description:

File write error.

Response message string:

"An error occurred while writing to file '%s'."

- `err_first`

Corresponding constant:

`mosek.rescode.err_first`

Description:

Invalid first.

Response message string:

"Invalid first."

- `err_firsti`

Corresponding constant:

`mosek.rescode.err_firsti`

Description:

Invalid firsti.

Response message string:

"'%d' is an invalid value for firsti."

- `err_firstj`

Corresponding constant:

`mosek.rescode.err_firstj`

Description:

Invalid firstj.

Response message string:

"'%d' is an invalid value for firstj."

- `err_flexlm`

Corresponding constant:

`mosek.rescode.err_flexlm`

Description:

The FLEXlm license manager reported an error.

Response message string:

“The FLEXlm license manager reported '%s'.”

- `err_huge_c`

Corresponding constant:

`mosek.rescode.err_huge_c`

Description:

A huge value in absolute size is specified for one c_j .

Response message string:

“A large value of %8.1e has been specified in cx for variable '%s' (%d).”

- `err_identical_tasks`

Corresponding constant:

`mosek.rescode.err_identical_tasks`

Description:

Some tasks related to this function call were identical. Unique tasks were expected.

Response message string:

“Some tasks related to this function call were identical. Unique tasks were expected.”

- `err_in_argument`

Corresponding constant:

`mosek.rescode.err_in_argument`

Description:

A function argument is incorrect.

Response message string:

“The argument '%s' is invalid.”

- `err_index`

Corresponding constant:

`mosek.rescode.err_index`

Description:

An index is out of range.

Response message string:

“An index is out of range.”

- `err_index_arr_is_too_large`

Corresponding constant:

`mosek.rescode.err_index_arr_is_too_large`

Description:

An index in an array argument is too large.

Response message string:

“The index value %d occurring in argument '%s[%d]' is too large(<%d).”

- `err_index_arr_is_too_small`

Corresponding constant:

`mosek.rescode.err_index_arr_is_too_small`

Description:

An index in an array argument is too small.

Response message string:

“The index value %d occurring in argument '%s[%d]' is too small(>=%d).”

- `err_index_is_too_large`

Corresponding constant:

`mosek.rescode.err_index_is_too_large`

Description:

An index in an argument is too large.

Response message string:

“The index value %d occurring in argument '%s' is too large.”

- `err_index_is_too_small`

Corresponding constant:

`mosek.rescode.err_index_is_too_small`

Description:

An index in an argument is too small.

Response message string:

“The index value %d occurring in argument '%s' is too small.”

- `err_inf_dou_index`

Corresponding constant:

`mosek.rescode.err_inf_dou_index`

Description:

A double information index is out of range for the specified type.

Response message string:

“The double information index %d is out of range.”

- `err_inf_dou_name`

Corresponding constant:

`mosek.rescode.err_inf_dou_name`

Description:

A double information name is invalid.

Response message string:

“The double information name '%s' is invalid.”

- `err_inf_int_index`

Corresponding constant:

`mosek.rescode.err_inf_int_index`

Description:

An integer information index is out of range for the specified type.

Response message string:

“The integer information index %d is out of range.”

- `err_inf_int_name`

Corresponding constant:

`mosek.rescode.err_inf_int_name`

Description:

A integer information name is invalid.

Response message string:

“The integer information name '%s' is invalid.”

- `err_inf_type`

Corresponding constant:

`mosek.rescode.err_inf_type`

Description:

The information type is invalid.

Response message string:

“The information type %d is invalid.”

- `err_infinite_bound`

Corresponding constant:

`mosek.rescode.err_infinite_bound`

Description:

A finite bound value is too large in absolute value.

Response message string:

“A finite bound value is too large in absolute value.”

- `err_internal`

Corresponding constant:

`mosek.rescode.err_internal`

Description:

An internal error occurred. Please report this problem.

Response message string:

“An internal error occurred '%s'.”

- `err_internal_test_failed`

Corresponding constant:

`mosek.rescode.err_internal_test_failed`

Description:

An internal unit test function failed.

Response message string:

“Internal unit test function failed.”

- `err_inv_aptre`

Corresponding constant:

`mosek.rescode.err_inv_aptre`

Description:

`aptre[j]` is strictly smaller than `aptrb[j]` for some `j`.

Response message string:

“`aptre` is strictly smaller than `aptrb` at position `%d`.”

- `err_inv_bk`

Corresponding constant:

`mosek.rescode.err_inv_bk`

Description:

Invalid bound key.

Response message string:

“`%d` is an invalid bound key.”

- `err_inv_bkc`

Corresponding constant:

`mosek.rescode.err_inv_bkc`

Description:

Invalid bound key is specified for a constraint.

Response message string:

“An invalid bound key for a constraint value of `%d` in argument ‘`%s`’ has been specified.”

- `err_inv_bkx`

Corresponding constant:

`mosek.rescode.err_inv_bkx`

Description:

An invalid bound key is specified for a variable.

Response message string:

“An invalid bound key for variable of value of `%d` in argument ‘`%s`’ has been specified.”

- `err_inv_cone_type`

Corresponding constant:

`mosek.rescode.err_inv_cone_type`

Description:

Invalid cone type code is encountered.

Response message string:

“’%d’ is an invalid cone type code.”

- `err_inv_cone_type_str`

Corresponding constant:

`mosek.rescode.err_inv_cone_type_str`

Description:

Invalid cone type string encountered.

Response message string:

“’%s’ is an invalid cone type string.”

- `err_inv_marki`

Corresponding constant:

`mosek.rescode.err_inv_marki`

Description:

Invalid value in marki.

Response message string:

“Invalid value in marki[%d].”

- `err_inv_markj`

Corresponding constant:

`mosek.rescode.err_inv_markj`

Description:

Invalid value in markj.

Response message string:

“Invalid value in markj[%d].”

- `err_inv_name_item`

Corresponding constant:

`mosek.rescode.err_inv_name_item`

Description:

An invalid name item code is used.

Response message string:

“’%d’ is an invalid name item code.”

- `err_inv_numi`

Corresponding constant:

`mosek.rescode.err_inv_numi`

Description:

Invalid numi.

Response message string:

“Invalid numi.”

- `err_inv_numj`

Corresponding constant:

`mosek.rescode.err_inv_numj`

Description:

Invalid numj.

Response message string:

“Invalid numj.”

- `err_inv_optimizer`

Corresponding constant:

`mosek.rescode.err_inv_optimizer`

Description:

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.

Response message string:

“An invalid optimizer (%d) has been chosen for the problem.”

- `err_inv_problem`

Corresponding constant:

`mosek.rescode.err_inv_problem`

Description:

Invalid problem type. Probably a nonconvex problem has been specified.

Response message string:

“Invalid problem type.”

- `err_inv_qcon_subi`

Corresponding constant:

`mosek.rescode.err_inv_qcon_subi`

Description:

Invalid value in qconsubi.

Response message string:

“Invalid value %d at qconsubi[%d].”

- `err_inv_qcon_subj`

Corresponding constant:

`mosek.rescode.err_inv_qcon_subj`

Description:

Invalid value in qcsubj.

Response message string:

“Invalid value %d at qcsubj[%d].”

- `err_inv_qcon_subk`

Corresponding constant:

`mosek.rescode.err_inv_qcon_subk`

Description:

Invalid value in qcsubk.

Response message string:

“Invalid value %d at qcsubk[%d].”

- `err_inv_qcon_val`

Corresponding constant:

`mosek.rescode.err_inv_qcon_val`

Description:

Invalid value in qcval.

Response message string:

“Invalid value %e at qcval[%d].”

- `err_inv_qobj_subi`

Corresponding constant:

`mosek.rescode.err_inv_qobj_subi`

Description:

Invalid value in qosubi.

Response message string:

“Invalid value %d at qosubi[%d].”

- `err_inv_qobj_subj`

Corresponding constant:

`mosek.rescode.err_inv_qobj_subj`

Description:

Invalid value in qosubj.

Response message string:

“Invalid value %d at qosubj[%d].”

- `err_inv_qobj_val`

Corresponding constant:

`mosek.rescode.err_inv_qobj_val`

Description:

Invalid value in qoval.

Response message string:

“Invalid value %e at qoval[%d].”

- `err_inv_sk`

Corresponding constant:

`mosek.rescode.err_inv_sk`

Description:

Invalid status key code.

Response message string:

“‘%d’ is an invalid status key code.”

- `err_inv_sk_str`

Corresponding constant:

`mosek.rescode.err_inv_sk_str`

Description:

Invalid status key string encountered.

Response message string:

“‘%s’ is an invalid status key string.”

- `err_inv_skc`

Corresponding constant:

`mosek.rescode.err_inv_skc`

Description:

Invalid value in skc.

Response message string:

“Invalid value at skc[%d].”

- `err_inv_skn`

Corresponding constant:

`mosek.rescode.err_inv_skn`

Description:

Invalid value in skn.

Response message string:

“Invalid value at skn[%d].”

- `err_inv_skx`

Corresponding constant:

`mosek.rescode.err_inv_skx`

Description:

Invalid value in skx.

Response message string:

“Invalid value at skx[%d].”

- `err_inv_var_type`

Corresponding constant:

`mosek.rescode.err_inv_var_type`

Description:

An invalid variable type is specified for a variable.

Response message string:

“An invalid type %d is specified for variable '%s' (%d) in argument '%s'.”

- `err_invalid_accmode`

Corresponding constant:

`mosek.rescode.err_invalid_accmode`

Description:

An invalid access mode is specified.

Response message string:

“%d is an invalid access mode is specified.”

- `err_invalid_ampl_stub`

Corresponding constant:

`mosek.rescode.err_invalid_ampl_stub`

Description:

Invalid AMPL stub.

Response message string:

“Invalid AMPL stub.”

- `err_invalid_branch_direction`

Corresponding constant:

`mosek.rescode.err_invalid_branch_direction`

Description:

An invalid branching direction is specified.

Response message string:

“%d is an invalid branching direction.”

- `err_invalid_branch_priority`

Corresponding constant:

`mosek.rescode.err_invalid_branch_priority`

Description:

An invalid branching priority is specified. It should be nonnegative.

Response message string:

“%d invalid branching priority is specified.”

- `err_invalid_compression`

Corresponding constant:

`mosek.rescode.err_invalid_compression`

Description:

Invalid compression type.

Response message string:

“%d is an invalid compression type.”

- `err_invalid_file_name`

Corresponding constant:

`mosek.rescode.err_invalid_file_name`

Description:

An invalid file name has been specified.

Response message string:

“'%s' is invalid file name.”

- `err_invalid_format_type`

Corresponding constant:

`mosek.rescode.err_invalid_format_type`

Description:

Invalid format type.

Response message string:

“%d is an invalid format type..”

- `err_invalid_iomode`

Corresponding constant:

`mosek.rescode.err_invalid_iomode`

Description:

Invalid io mode.

Response message string:

“%d is an io mode.”

- `err_invalid_mbt_file`

Corresponding constant:

`mosek.rescode.err_invalid_mbt_file`

Description:

A MOSEK binary task file is invalid.

Response message string:

“The binary task file is invalid.”

- `err_invalid_name_in_sol_file`

Corresponding constant:

`mosek.rescode.err_invalid_name_in_sol_file`

Description:

An invalid name occurred in a solution file.

Response message string:

"The name '%s' is an invalid name."

- `err_invalid_obj_name`

Corresponding constant:

`mosek.rescode.err_invalid_obj_name`

Description:

An invalid objective name is specified.

Response message string:

"'%s' is an invalid objective name is specified."

- `err_invalid_objective_sense`

Corresponding constant:

`mosek.rescode.err_invalid_objective_sense`

Description:

An invalid objective sense is specified.

Response message string:

"%s is an invalid objective sense code."

- `err_invalid_sol_file_name`

Corresponding constant:

`mosek.rescode.err_invalid_sol_file_name`

Description:

An invalid file name has been specified.

Response message string:

"'%s' is invalid solution file name."

- `err_invalid_stream`

Corresponding constant:

`mosek.rescode.err_invalid_stream`

Description:

An invalid stream is referenced.

Response message string:

"%d is an invalid stream."

- `err_invalid_task`

Corresponding constant:

`mosek.rescode.err_invalid_task`

Description:

The task is invalid.

Response message string:

“The task is invalid.”

- `err_invalid_utf8`

Corresponding constant:

`mosek.rescode.err_invalid_utf8`

Description:

An invalid UTF8 string is encountered.

Response message string:

“An invalid UTF8 string is encountered.”

- `err_invalid_wchar`

Corresponding constant:

`mosek.rescode.err_invalid_wchar`

Description:

An invalid wchar string is encountered.

Response message string:

“An invalid wchar string is encountered.”

- `err_last`

Corresponding constant:

`mosek.rescode.err_last`

Description:

Invalid last.

Response message string:

“Invalid last.”

- `err_lasti`

Corresponding constant:

`mosek.rescode.err_lasti`

Description:

Invalid lasti.

Response message string:

“’%d’ is an invalid value for lasti.”

- `err_lastj`

Corresponding constant:

`mosek.rescode.err_lastj`

Description:

Invalid lastj.

Response message string:

“’%d’ is an invalid value for lastj.”

- `err_license`

Corresponding constant:

`mosek.rescode.err_license`

Description:

Invalid license.

Response message string:

"Invalid license."

- `err_license_cannot_allocate`

Corresponding constant:

`mosek.rescode.err_license_cannot_allocate`

Description:

The license system cannot allocate the memory required.

Response message string:

"The license system cannot allocate the memory required."

- `err_license_cannot_connect`

Corresponding constant:

`mosek.rescode.err_license_cannot_connect`

Description:

MOSEK cannot connect to the license server. Most likely the license server is not up and running.

Response message string:

"MOSEK cannot connect to the license server."

- `err_license_expired`

Corresponding constant:

`mosek.rescode.err_license_expired`

Description:

The license has expired.

Response message string:

"The license has expired."

- `err_license_feature`

Corresponding constant:

`mosek.rescode.err_license_feature`

Description:

A requested feature is not available in the license file(s). Most likely due to an incorrect license system setup.

Response message string:

"The feature '%s' is not in license file. Consult the license manager error message."

- `err_license_invalid_hostid`

Corresponding constant:

`mosek.rescode.err_license_invalid_hostid`

Description:

The host ID specified in the license file does not match the host ID of the computer.

Response message string:

“The host ID specified in the license file does not match the host ID of the computer.”

- `err_license_max`

Corresponding constant:

`mosek.rescode.err_license_max`

Description:

Maximum number of licenses is reached.

Response message string:

“Maximum number of licenses is reached for feature '%s'.”

- `err_license_moseklm_daemon`

Corresponding constant:

`mosek.rescode.err_license_moseklm_daemon`

Description:

The MOSEKLM license manager daemon is not up and running.

Response message string:

“The MOSEKLM license manager daemon is not up and running.”

- `err_license_server`

Corresponding constant:

`mosek.rescode.err_license_server`

Description:

The license server is not responding.

Response message string:

“The license server is not responding.”

- `err_license_server_version`

Corresponding constant:

`mosek.rescode.err_license_server_version`

Description:

The version specified in the checkout request is greater than the highest version number the daemon supports.

Response message string:

“License server system does not support this version '%d' of feature '%s'.”

- `err_license_version`

Corresponding constant:

`mosek.rescode.err_license_version`

Description:

The license is valid for another version of MOSEK.

Response message string:

“Version '%s' of feature '%s' is not supported by the license file.”

- `err_link_file_dll`

Corresponding constant:

`mosek.rescode.err_link_file_dll`

Description:

A file cannot be linked to a stream in the DLL version.

Response message string:

“A file cannot be linked to a stream in the DLL version.”

- `err_lp_dup_slack_name`

Corresponding constant:

`mosek.rescode.err_lp_dup_slack_name`

Description:

The name of the slack variable added to a ranged constraint already exists.

Response message string:

“Slack variable name '%s' in constraint '%s' id defined already.”

- `err_lp_empty`

Corresponding constant:

`mosek.rescode.err_lp_empty`

Description:

The problem cannot be written to an LP formatted file.

Response message string:

“A problem with no variables or constraints cannot be written to a LP formatted file.”

- `err_lp_file_format`

Corresponding constant:

`mosek.rescode.err_lp_file_format`

Description:

Syntax error in an LP file.

Response message string:

“Syntax error in an LP file at (%d:%d).”

- `err_lp_format`

Corresponding constant:

`mosek.rescode.err_lp_format`

Description:

Syntax error in an LP file.

Response message string:

“Syntax error in an LP file at line number: %d.”

- `err_lp_free_constraint`

Corresponding constant:

`mosek.rescode.err_lp_free_constraint`

Description:

Free constraints cannot be written in LP file format.

Response message string:

“Free constraints cannot be written in LP file format.”

- `err_lp_incompatible`

Corresponding constant:

`mosek.rescode.err_lp_incompatible`

Description:

The problem cannot be written to an LP formatted file.

Response message string:

“A problem of type '%s' cannot be written to a LP formatted file.”

- `err_lp_invalid_var_name`

Corresponding constant:

`mosek.rescode.err_lp_invalid_var_name`

Description:

A variable name is invalid when used in an LP formatted file.

Response message string:

“The variable name '%s' cannot be written to an LP formatted file.”

- `err_lp_write_conic_problem`

Corresponding constant:

`mosek.rescode.err_lp_write_conic_problem`

Description:

The problem contains cones that cannot be written to an LP formatted file.

Response message string:

“A problem of type '%s' contains cones that cannot be written to an LP formatted file.”

- `err_lp_write_geco_problem`

Corresponding constant:

`mosek.rescode.err_lp_write_geco_problem`

Description:

The problem contains general convex terms that cannot be written to an LP formatted file.

Response message string:

“A problem of type '%s' contains general convex terms that cannot be written to an LP formatted file.”

- `err_lu_max_num_tries`

Corresponding constant:

`mosek.rescode.err_lu_max_num_tries`

Description:

Could not compute the LU factors of the matrix within the maximum number of allowed tries.

Response message string:

“Could not compute the LU factors of the matrix within the maximum number of allowed tries.”

- `err_maxnumanz`

Corresponding constant:

`mosek.rescode.err_maxnumanz`

Description:

The maximum number of non-zeros specified for A is smaller than the number of non-zeros in the current A .

Response message string:

“Too small maximum number of non-zeros in A specified.”

- `err_maxnumcon`

Corresponding constant:

`mosek.rescode.err_maxnumcon`

Description:

The maximum number of constraints specified is smaller than the number of constraints in the task.

Response message string:

“Maximum number of constraints of '%d' is smaller than the number of constraints '%d'.”

- `err_maxnumcone`

Corresponding constant:

`mosek.rescode.err_maxnumcone`

Description:

The value specified for `maxnumcone` is too small.

Response message string:

"The value %d specified for maxnumcone is too small."

- `err_maxnumqnz`

Corresponding constant:

`mosek.rescode.err_maxnumqnz`

Description:

The maximum number of non-zeros specified for the Q matrices is smaller than the number of non-zeros in the current Q matrices.

Response message string:

"Too small maximum number of non-zeros for the Q matrices is specified."

- `err_maxnumvar`

Corresponding constant:

`mosek.rescode.err_maxnumvar`

Description:

The maximum number of variables specified is smaller than the number of variables in the task.

Response message string:

"Too small maximum number of variables %d is specified. Currently, the number of variables is %d."

- `err_mbt_incompatible`

Corresponding constant:

`mosek.rescode.err_mbt_incompatible`

Description:

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.

Response message string:

"The MBT file is incompatible with this platform."

- `err_mio_no_optimizer`

Corresponding constant:

`mosek.rescode.err_mio_no_optimizer`

Description:

No optimizer is available for the current class of integer optimization problems.

Response message string:

"No integer optimizer is available for the optimization problem."

- `err_mio_not_loaded`

Corresponding constant:

`mosek.rescode.err_mio_not_loaded`

Description:

The mixed-integer optimizer is not loaded.

Response message string:

“The mixed-integer optimizer is not loaded.”

- `err_missing_license_file`

Corresponding constant:

`mosek.rescode.err_missing_license_file`

Description:

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.

Response message string:

“A license file is missing. Set `MOSEKLM_LICENSE_FILE` to point to your license file.”

- `err_mixed_problem`

Corresponding constant:

`mosek.rescode.err_mixed_problem`

Description:

The problem contains both conic and nonlinear constraints.

Response message string:

“The problem contains both conic and nonlinear constraints.”

- `err_mps_cone_overlap`

Corresponding constant:

`mosek.rescode.err_mps_cone_overlap`

Description:

A variable is specified to be a member of several cones.

Response message string:

“Variable '%s' is specified to be a member of CSECTION '%s' and CSECTION '%s'.”

- `err_mps_cone_repeat`

Corresponding constant:

`mosek.rescode.err_mps_cone_repeat`

Description:

A variable is repeated within the CSECTION.

Response message string:

“Variable '%s' is repeated in CSECTION '%s'.”

- `err_mps_cone_type`

Corresponding constant:

`mosek.rescode.err_mps_cone_type`

Description:

Invalid cone type specified in a CSECTION.

Response message string:

“'%s' is an invalid cone type in a CSECTION.”

- `err_mps_file`

Corresponding constant:

`mosek.rescode.err_mps_file`

Description:

An error occurred while reading an MPS file.

Response message string:

“An error occurred while reading an MPS file.”

- `err_mps_inv_bound_key`

Corresponding constant:

`mosek.rescode.err_mps_inv_bound_key`

Description:

An invalid bound key occurred in an MPS file.

Response message string:

“'%s' is an invalid bound key.”

- `err_mps_inv_con_key`

Corresponding constant:

`mosek.rescode.err_mps_inv_con_key`

Description:

An invalid constraint key occurred in an MPS file.

Response message string:

“'%s' is an invalid constraint key for constraint '%s'.”

- `err_mps_inv_field`

Corresponding constant:

`mosek.rescode.err_mps_inv_field`

Description:

A field in the MPS file is invalid. Probably it is too wide.

Response message string:

“Field number %d is invalid.”

- `err_mps_inv_marker`

Corresponding constant:

`mosek.rescode.err_mps_inv_marker`

Description:

An invalid marker has been specified in the MPS file.

Response message string:

“An invalid marker has been specified in the MPS file.”

- `err_mps_inv_sec_name`

Corresponding constant:

`mosek.rescode.err_mps_inv_sec_name`

Description:

An invalid section name occurred in an MPS file.

Response message string:

“An invalid section name was used.”

- `err_mps_inv_sec_order`

Corresponding constant:

`mosek.rescode.err_mps_inv_sec_order`

Description:

The sections in the MPS data file are not in the correct order.

Response message string:

“Section '%s' was not expected.”

- `err_mps_invalid_obj_name`

Corresponding constant:

`mosek.rescode.err_mps_invalid_obj_name`

Description:

An invalid objective name is specified.

Response message string:

“'%s' is an invalid objective name is specified.”

- `err_mps_invalid_objsense`

Corresponding constant:

`mosek.rescode.err_mps_invalid_objsense`

Description:

An invalid objective sense is specified.

Response message string:

“'%s' is an invalid objective sense.”

- `err_mps_mul_con_name`

Corresponding constant:

`mosek.rescode.err_mps_mul_con_name`

Description:

A constraint name was specified multiple times in the ROWS section.

Response message string:

“The constraint name '%s' was specified multiple times in the ROWS section.”

- `err_mps_mul_csec`

Corresponding constant:

`mosek.rescode.err_mps_mul_csec`

Description:

Multiple CSECTIONs are given the same name.

Response message string:

“Multiple CSECTIONs are given the name '%s'.”

- `err_mps_mul_qobj`

Corresponding constant:

`mosek.rescode.err_mps_mul_qobj`

Description:

The Q term in the objective is specified multiple times in the MPS data file.

Response message string:

“The Q term in the objective is specified multiple times.”

- `err_mps_mul_qsec`

Corresponding constant:

`mosek.rescode.err_mps_mul_qsec`

Description:

Multiple QSECTIONs are specified for a constraint in the MPS data file.

Response message string:

“Multiple QSECTIONs are specified for a constraint '%s'.”

- `err_mps_no_objective`

Corresponding constant:

`mosek.rescode.err_mps_no_objective`

Description:

No objective is defined in an MPS file.

Response message string:

“No objective was defined.”

- `err_mps_null_con_name`

Corresponding constant:

`mosek.rescode.err_mps_null_con_name`

Description:

An empty constraint name is used in an MPS file.

Response message string:

“An empty constraint name is used in an MPS file.”

- `err_mps_null_var_name`

Corresponding constant:

mosek.rescode.err_mps_null_var_name

Description:

An empty variable name is used in an MPS file.

Response message string:

“An empty variable name is used in an MPS file.”

- `err_mpsSplittedVar`

Corresponding constant:

mosek.rescode.err_mpsSplittedVar

Description:

A variable is split in an MPS data file.

Response message string:

“The variable '%s' was split.”

- `err_mpsTabInField2`

Corresponding constant:

mosek.rescode.err_mpsTabInField2

Description:

A tab char occurred in field 2.

Response message string:

“A tab char occurred in field 2.”

- `err_mpsTabInField3`

Corresponding constant:

mosek.rescode.err_mpsTabInField3

Description:

A tab char occurred in field 3.

Response message string:

“A tab char occurred in field 3.”

- `err_mpsTabInField5`

Corresponding constant:

mosek.rescode.err_mpsTabInField5

Description:

A tab char occurred in field 5.

Response message string:

“A tab char occurred in field 5.”

- `err_mpsUndefConName`

Corresponding constant:

mosek.rescode.err_mpsUndefConName

Description:

An undefined constraint name occurred in an MPS file.

Response message string:

“‘%s’ is an undefined constraint name.”

- `err_mps_undef_var_name`

Corresponding constant:

`mosek.rescode.err_mps_undef_var_name`

Description:

An undefined variable name occurred in an MPS file.

Response message string:

“‘%s’ is an undefined variable name.”

- `err_mul_a_element`

Corresponding constant:

`mosek.rescode.err_mul_a_element`

Description:

An element in A is defined multiple times.

Response message string:

“Multiple elements in row %d of A at column %d.”

- `err_name_max_len`

Corresponding constant:

`mosek.rescode.err_name_max_len`

Description:

A name is longer than the buffer that is supposed to hold it.

Response message string:

“A name(‘%s’) of length %d is longer than the buffer of length %d that is supposed to hold it.”

- `err_nan_in_blc`

Corresponding constant:

`mosek.rescode.err_nan_in_blc`

Description:

l^c contains an invalid floating point value, i.e. a NaN.

Response message string:

“The bound vector blc contains an invalid value for constraint ‘%s’ (%d).”

- `err_nan_in_blx`

Corresponding constant:

`mosek.rescode.err_nan_in_blx`

Description:

l^x contains an invalid floating point value, i.e. a NaN.

Response message string:

“The bound vector blx contains an invalid value for variable '%s' (%d).”

- `err_nan_in_buc`

Corresponding constant:

`mosek.rescode.err_nan_in_buc`

Description:

u^c contains an invalid floating point value, i.e. a NaN.

Response message string:

“The bound vector buc contains an invalid value for constraint '%s' (%d).”

- `err_nan_in_bux`

Corresponding constant:

`mosek.rescode.err_nan_in_bux`

Description:

u^x contains an invalid floating point value, i.e. a NaN.

Response message string:

“The bound vector bux contains an invalid value for variable '%s' (%d).”

- `err_nan_in_c`

Corresponding constant:

`mosek.rescode.err_nan_in_c`

Description:

c contains an invalid floating point value, i.e. a NaN.

Response message string:

“The objective vector c contains an invalid value for variable '%s' (%d).”

- `err_nan_in_double_data`

Corresponding constant:

`mosek.rescode.err_nan_in_double_data`

Description:

An invalid floating point value was used in some double data.

Response message string:

“The parameter '%s' contained an invalid floating value.”

- `err_negative_append`

Corresponding constant:

`mosek.rescode.err_negative_append`

Description:

Cannot append a negative number.

Response message string:

“Cannot append a negative number of %d.”

- `err_negative_surplus`

Corresponding constant:

`mosek.rescode.err_negative_surplus`

Description:

Negative surplus.

Response message string:

“Negative surplus.”

- `err_newer_dll`

Corresponding constant:

`mosek.rescode.err_newer_dll`

Description:

The dynamic link library is newer than the specified version.

Response message string:

“The dynamic link library version %d.%d.%d.%d is newer than version %d.%d.%d.%d.”

- `err_no_basis_sol`

Corresponding constant:

`mosek.rescode.err_no_basis_sol`

Description:

No basic solution is defined.

Response message string:

“No basic solution is defined.”

- `err_no_dual_for_itg_sol`

Corresponding constant:

`mosek.rescode.err_no_dual_for_itg_sol`

Description:

No dual information is available for the integer solution.

Response message string:

“No dual information is available for the integer solution.”

- `err_no_dual_infeas_cer`

Corresponding constant:

`mosek.rescode.err_no_dual_infeas_cer`

Description:

A certificate of infeasibility is not available.

Response message string:

“A certificate of dual infeasibility is not available.”

- `err_no_init_env`

Corresponding constant:

`mosek.rescode.err_no_init_env`

Description:

env is not initialized.

Response message string:

“Environment is not initialized.”

- `err_no_optimizer_var_type`

Corresponding constant:

`mosek.rescode.err_no_optimizer_var_type`

Description:

No optimizer is available for this class of optimization problems.

Response message string:

“No optimizer is available for optimization problems containing variables of type '%s'.”

- `err_no_primal_infeas_cer`

Corresponding constant:

`mosek.rescode.err_no_primal_infeas_cer`

Description:

A certificate of primal infeasibility is not available.

Response message string:

“A certificate of primal infeasibility is not available.”

- `err_no_solution_in_callback`

Corresponding constant:

`mosek.rescode.err_no_solution_in_callback`

Description:

The required solution is not available.

Response message string:

“The required solution is not available.”

- `err_nonconvex`

Corresponding constant:

`mosek.rescode.err_nonconvex`

Description:

The optimization problem is nonconvex.

Response message string:

“The optimization problem is nonconvex.”

- `err_nonlinear_equality`

Corresponding constant:

`mosek.rescode.err_nonlinear_equality`

Description:

The model contains a nonlinear equality which defines a nonconvex set.

Response message string:

“Non convex model detected. Constraint '%s'(%d) is a nonlinear equality.”

- `err_nonlinear_ranged`

Corresponding constant:

`mosek.rescode.err_nonlinear_ranged`

Description:

The model contains a nonlinear ranged constraint which by definition defines a nonconvex set.

Response message string:

“Constraint '%s'(%d)' is nonlinear and ranged constraint i.e. it has finite lower and upper bound.”

- `err_nr_arguments`

Corresponding constant:

`mosek.rescode.err_nr_arguments`

Description:

Incorrect number of function arguments.

Response message string:

“Incorrect number of %s arguments. Got %d expected %d.”

- `err_null_env`

Corresponding constant:

`mosek.rescode.err_null_env`

Description:

`env` is a NULL pointer.

Response message string:

“`env` is a NULL pointer.”

- `err_null_name`

Corresponding constant:

`mosek.rescode.err_null_name`

Description:

An all blank name has been specified.

Response message string:

“An all blank name has been specified.”

- `err_null_pointer`

Corresponding constant:

`mosek.rescode.err_null_pointer`

Description:

An argument to a function is unexpectedly a NULL pointer.

Response message string:

“Argument ‘%s’ is unexpectedly a NULL pointer.”

- `err_null_task`

Corresponding constant:

`mosek.rescode.err_null_task`

Description:

task is a NULL pointer.

Response message string:

“task is a NULL pointer.”

- `err_numconlim`

Corresponding constant:

`mosek.rescode.err_numconlim`

Description:

Maximum number of constraints limit is exceeded.

Response message string:

“Maximum number of constraints limit is exceeded.”

- `err_numvarlim`

Corresponding constant:

`mosek.rescode.err_numvarlim`

Description:

Maximum number of variables limit is exceeded.

Response message string:

“Maximum number of variables limit is exceeded.”

- `err_obj_q_not_nsd`

Corresponding constant:

`mosek.rescode.err_obj_q_not_nsd`

Description:

The quadratic coefficient matrix in the objective is not negative semi-definite as expected for a maximization problem.

Response message string:

“The quadratic coefficient matrix in the objective is not negative semi-definite as expected for a maximization problem.”

- `err_obj_q_not_psd`

Corresponding constant:

`mosek.rescode.err_obj_q_not_psd`

Description:

The quadratic coefficient matrix in the objective is not positive semi-definite as expected for a minimization problem.

Response message string:

“The quadratic coefficient matrix in the objective is not positive semi-definite as expected for a minimization problem.”

- `err_objective_range`

Corresponding constant:

`mosek.rescode.err_objective_range`

Description:

Empty objective range.

Response message string:

“Empty objective range.”

- `err_older_dll`

Corresponding constant:

`mosek.rescode.err_older_dll`

Description:

The dynamic link library is older than the specified version.

Response message string:

“The dynamic link library version %d.%d.%d.%d is older than expected version %d.%d.%d.%d.”

- `err_open_dl`

Corresponding constant:

`mosek.rescode.err_open_dl`

Description:

A dynamic link library could not be opened.

Response message string:

“A dynamic link library '%s' could not be opened.”

- `err_opf_format`

Corresponding constant:

`mosek.rescode.err_opf_format`

Description:

Syntax error in an OPF file

Response message string:

“Syntax error in an OPF file at line number: %d.”

- `err_optimizer_license`

Corresponding constant:

`mosek.rescode.err_optimizer_license`

Description:

The optimizer required is not licensed.

Response message string:

"The optimizer required is not licensed."

- `err_ord_invalid`

Corresponding constant:

`mosek.rescode.err_ord_invalid`

Description:

Invalid content in branch ordering file.

Response message string:

"Invalid content in branch ordering file"

- `err_ord_invalid_branch_dir`

Corresponding constant:

`mosek.rescode.err_ord_invalid_branch_dir`

Description:

An invalid branch direction key is specified.

Response message string:

"'%s' is an invalid branch direction key is specified."

- `err_param_index`

Corresponding constant:

`mosek.rescode.err_param_index`

Description:

Parameter index is out of range.

Response message string:

"The parameter index %d is invalid for a parameter of type %s."

- `err_param_is_too_large`

Corresponding constant:

`mosek.rescode.err_param_is_too_large`

Description:

The parameter value is too large.

Response message string:

"The parameter value %s is too large for parameter '%s'."

- `err_param_is_too_small`

Corresponding constant:

`mosek.rescode.err_param_is_too_small`

Description:

The parameter value is too small.

Response message string:

“The parameter value %s is too small for parameter '%s’.”

- `err_param_name`

Corresponding constant:

`mosek.rescode.err_param_name`

Description:

The parameter name is not correct.

Response message string:

“The parameter name '%s' is invalid.”

- `err_param_name_dou`

Corresponding constant:

`mosek.rescode.err_param_name_dou`

Description:

The parameter name is not correct for a double parameter.

Response message string:

“The parameter name '%s' is invalid for a double parameter.”

- `err_param_name_int`

Corresponding constant:

`mosek.rescode.err_param_name_int`

Description:

The parameter name is not correct for an integer parameter.

Response message string:

“The parameter name '%s' is invalid for an int parameter.”

- `err_param_name_str`

Corresponding constant:

`mosek.rescode.err_param_name_str`

Description:

The parameter name is not correct for a string parameter.

Response message string:

“The parameter name '%s' is invalid for a string parameter.”

- `err_param_type`

Corresponding constant:

`mosek.rescode.err_param_type`

Description:

The parameter type is invalid.

Response message string:

“The parameter type %d is invalid.”

- `err_param_value_str`

Corresponding constant:

`mosek.rescode.err_param_value_str`

Description:

The parameter value string is incorrect.

Response message string:

“The parameter value string '%s' for parameter %s is incorrect.”

- `err_platform_not_licensed`

Corresponding constant:

`mosek.rescode.err_platform_not_licensed`

Description:

A requested license feature is not available for the required platform.

Response message string:

“No license feature '%s' for the required platform is available.”

- `err_postsolve`

Corresponding constant:

`mosek.rescode.err_postsolve`

Description:

An error occurred during the postsolve. Please contact MOSEK support.

Response message string:

“An error occurred during the postsolve.”

- `err_pro_item`

Corresponding constant:

`mosek.rescode.err_pro_item`

Description:

An invalid problem is used.

Response message string:

“'%d' is an invalid problem item.”

- `err_prob_license`

Corresponding constant:

`mosek.rescode.err_prob_license`

Description:

The software is not licensed to solve the problem.

Response message string:

“The software is not licensed to solve the problem.”

- `err_qcon_subi_too_large`

Corresponding constant:

`mosek.rescode.err_qcon_subi_too_large`

Description:

Invalid value in `qconsubi`.

Response message string:

“Invalid value %d at `qconsubi[%d]`. It should be < %d.”

- `err_qcon_subi_too_small`

Corresponding constant:

`mosek.rescode.err_qcon_subi_too_small`

Description:

Invalid value in `qconsubi`.

Response message string:

“Invalid value %d at `qconsubi[%d]`. It should be >= %d.”

- `err_qcon_upper_triangle`

Corresponding constant:

`mosek.rescode.err_qcon_upper_triangle`

Description:

An element in the upper triangle of a Q^k is specified. Only elements in the lower triangle should be specified.

Response message string:

“The element `q[%d,%d]` in the upper triangle of the quadratic term in the %dth constraint is specified.”

- `err_qobj_upper_triangle`

Corresponding constant:

`mosek.rescode.err_qobj_upper_triangle`

Description:

An element in the upper triangle of Q^o is specified. Only elements in the lower triangle should be specified.

Response message string:

“The element `q[%d,%d]` in the upper triangle of the quadratic term in the objective is specified.”

- `err_read_format`

Corresponding constant:

`mosek.rescode.err_read_format`

Description:

The specified format cannot be read.

Response message string:

“The specified format cannot be read. The format code is %d.”

- `err_read_lp_nonexisting_name`

Corresponding constant:

`mosek.rescode.err_read_lp_nonexisting_name`

Description:

A variable never occurred in objective or constraints.

Response message string:

“The variable name '%s' did not occur in objective or constraints.”

- `err_remove_cone_variable`

Corresponding constant:

`mosek.rescode.err_remove_cone_variable`

Description:

A variable cannot be removed because it will make a cone invalid.

Response message string:

“If variable %d ('%s') is removed, then cone %d ('%s') will be invalid.”

- `err_sen_bound_invalid_lo`

Corresponding constant:

`mosek.rescode.err_sen_bound_invalid_lo`

Description:

Analysis of lower bound requested for an index, where no lower bound exists.

Response message string:

“No lower bound for index '%d' given in line %d.”

- `err_sen_bound_invalid_up`

Corresponding constant:

`mosek.rescode.err_sen_bound_invalid_up`

Description:

Analysis of upper bound requested for an index, where no upper bound exists.

Response message string:

“No upper bound for index '%d' given in line %d.”

- `err_sen_format`

Corresponding constant:

`mosek.rescode.err_sen_format`

Description:

Syntax error in sensitivity analysis file.

Response message string:

“Syntax error in sensitivity analysis file at line number: %d. %s”

- `err_sen_index_invalid`

Corresponding constant:

`mosek.rescode.err_sen_index_invalid`

Description:

Invalid range given in the sensitivity file.

Response message string:

“The index range %d-%d in line %d is invalid.”

- `err_sen_index_range`

Corresponding constant:

`mosek.rescode.err_sen_index_range`

Description:

Index out of range in the sensitivity analysis file.

Response message string:

“Index '%d' out of range at line %d.”

- `err_sen_invalid_regexp`

Corresponding constant:

`mosek.rescode.err_sen_invalid_regexp`

Description:

Syntax error in regexp or regexp longer than 1024.

Response message string:

“Syntax error in regexp on line %d: %s.”

- `err_sen_numerical`

Corresponding constant:

`mosek.rescode.err_sen_numerical`

Description:

Numerical difficulties encountered performing the sensitivity analysis.

Response message string:

“Numerical difficulties encountered performing the sensitivity analysis.”

- `err_sen_solution_status`

Corresponding constant:

`mosek.rescode.err_sen_solution_status`

Description:

No optimal solution found to the original problem given for sensitivity analysis.

Response message string:

“No optimal solution found to the original problem given for sensitivity analysis.
Solution status = %d.”

- `err_sen_undef_name`

Corresponding constant:

`mosek.rescode.err_sen_undef_name`

Description:

An undefined name was encountered in the sensitivity analysis file.

Response message string:

“Name '%s' on line %d not defined.”

- `err_size_license`

Corresponding constant:

`mosek.rescode.err_size_license`

Description:

The problem is bigger than the license.

Response message string:

“The problem is bigger than the license.”

- `err_size_license_con`

Corresponding constant:

`mosek.rescode.err_size_license_con`

Description:

The problem has too many constraints to be solved with the available license.

Response message string:

“The problem has %d constraint(s) but the license allows only %d constraint(s)
for feature '%s'.”

- `err_size_license_intvar`

Corresponding constant:

`mosek.rescode.err_size_license_intvar`

Description:

The problem contains too many integer variables to be solved with the available license.

Response message string:

“The problem contains %d integer variable(s) but the license allows only %d integer
variable(s) for feature '%s'.”

- `err_size_license_var`

Corresponding constant:

`mosek.rescode.err_size_license_var`

Description:

The problem has too many variables to be solved with the available license.

Response message string:

“The problem has %d variable(s) but the license allows only %d variable(s) for feature '%s'.”

- `err_sol_file_number`

Corresponding constant:

`mosek.rescode.err_sol_file_number`

Description:

An invalid number is specified in a solution file.

Response message string:

“An invalid number '%s' is specified in a solution file.”

- `err_solitem`

Corresponding constant:

`mosek.rescode.err_solitem`

Description:

The solution item number `solitem` is invalid. Please note `mosek.solitem.snx` is invalid for the basic solution.

Response message string:

“%d is not a valid solution item code for solution %d.”

- `err_solver_probtype`

Corresponding constant:

`mosek.rescode.err_solver_probtype`

Description:

Problem type does not match the chosen optimizer.

Response message string:

“Problem type does not match the chosen optimizer.”

- `err_space`

Corresponding constant:

`mosek.rescode.err_space`

Description:

Out of space.

Response message string:

“Out of space.”

- `err_space_leaking`

Corresponding constant:

`mosek.rescode.err_space_leaking`

Description:

MOSEK is leaking memory. This can be due to either an incorrect use of MOSEK or a bug.

Response message string:

“MOSEK is leaking memory.”

- `err_space_no_info`

Corresponding constant:

`mosek.rescode.err_space_no_info`

Description:

No available information about the space usage.

Response message string:

“No available information about the space usage.”

- `err_thread_cond_init`

Corresponding constant:

`mosek.rescode.err_thread_cond_init`

Description:

Could not initialize a condition.

Response message string:

“Could not initialize a condition.”

- `err_thread_create`

Corresponding constant:

`mosek.rescode.err_thread_create`

Description:

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.

Response message string:

“Could not create a thread. System error code: %d”

- `err_thread_mutex_init`

Corresponding constant:

`mosek.rescode.err_thread_mutex_init`

Description:

Could not initialize a mutex.

Response message string:

“Could not initialize a mutex.”

- `err_thread_mutex_lock`

Corresponding constant:

`mosek.rescode.err_thread_mutex_lock`

Description:

Could not lock a mutex.

Response message string:

“Could not lock a mutex.”

- `err_thread_mutex_unlock`

Corresponding constant:

`mosek.rescode.err_thread_mutex_unlock`

Description:

Could not unlock a mutex.

Response message string:

“Could not unlock a mutex.”

- `err_too_small_maxnumanz`

Corresponding constant:

`mosek.rescode.err_too_small_maxnumanz`

Description:

Maximum number of non-zeros allowed in A is too small.

Response message string:

“Maximum number of non-zeros allowed in A is too small. %d is required.”

- `err_unb_step_size`

Corresponding constant:

`mosek.rescode.err_unb_step_size`

Description:

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.

Response message string:

“A step-size in an optimizer was unexpectedly unbounded.”

- `err_undef_solution`

Corresponding constant:

`mosek.rescode.err_undef_solution`

Description:

The required solution is not defined.

Response message string:

“The solution with code %d is not defined.”

- `err_undefined_objective_sense`

Corresponding constant:

`mosek.rescode.err_undefined_objective_sense`

Description:

The objective sense has not been specified before the optimization.

Response message string:

"The objective sense has not been specified before the optimization."

- `err_unknown`

Corresponding constant:

`mosek.rescode.err_unknown`

Description:

Unknown error.

Response message string:

"Unknown error."

- `err_user_func_ret`

Corresponding constant:

`mosek.rescode.err_user_func_ret`

Description:

An user function reported an error.

Response message string:

"An user function returned a non-zero error code %d."

- `err_user_func_ret_data`

Corresponding constant:

`mosek.rescode.err_user_func_ret_data`

Description:

An user function returned invalid data.

Response message string:

"An user function returned invalid data for '%s'."

- `err_user_nlo_eval`

Corresponding constant:

`mosek.rescode.err_user_nlo_eval`

Description:

The user-defined nonlinear function reported an error.

Response message string:

"The user-defined nonlinear function reported the error '%s'."

- `err_user_nlo_eval_hessubi`

Corresponding constant:

`mosek.rescode.err_user_nlo_eval_hessubi`

Description:

The user-defined nonlinear function reported an invalid subscript in the Hessian.

Response message string:

“The user-defined nonlinear function reported the invalid hessubi[%d]: %d.”

- `err_user_nlo_eval_hessubj`

Corresponding constant:

`mosek.rescode.err_user_nlo_eval_hessubj`

Description:

The user-defined nonlinear function reported an invalid subscript in the Hessian.

Response message string:

“The user-defined nonlinear function reported the invalid subscript hessubj[%d]: %d.”

- `err_user_nlo_func`

Corresponding constant:

`mosek.rescode.err_user_nlo_func`

Description:

The user-defined nonlinear function reported an error.

Response message string:

“The user-defined nonlinear function reported an error.”

- `err_whichitem_not_allowed`

Corresponding constant:

`mosek.rescode.err_whichitem_not_allowed`

Description:

`whichitem` is unacceptable.

Response message string:

“%d is an unacceptable `whichitem`.”

- `err_whichsol`

Corresponding constant:

`mosek.rescode.err_whichsol`

Description:

The solution defined by `compwhichsol` does not exist.

Response message string:

“%d is not a valid solution code.”

- `err_write_lp_format`

Corresponding constant:

`mosek.rescode.err_write_lp_format`

Description:

Problem cannot be written as an LP file.

Response message string:

“Problem cannot be written as an LP file because of: %s.”

- `err_write_lp_non_unique_name`

Corresponding constant:

`mosek.rescode.err_write_lp_non_unique_name`

Description:

An auto-generated name is not unique.

Response message string:

“The auto-generated name '%s' is not unique.”

- `err_write_mps_invalid_name`

Corresponding constant:

`mosek.rescode.err_write_mps_invalid_name`

Description:

An invalid name is created while writing an MPS file. Usually this will make the MPS file unreadable.

Response message string:

“The name '%s' is not a valid MPS name.”

- `err_write_opf_invalid_var_name`

Corresponding constant:

`mosek.rescode.err_write_opf_invalid_var_name`

Description:

Empty variable names cannot be written to OPF files.

Response message string:

“Name of variable index %d is empty and cannot be written to an OPF file.”

- `err_xml_invalid_problem_type`

Corresponding constant:

`mosek.rescode.err_xml_invalid_problem_type`

Description:

The problem type is not supported by the XML format.

Response message string:

“The problem type %s is not supported by the XML format.”

- `err_y_is_undefined`

Corresponding constant:

`mosek.rescode.err_y_is_undefined`

Description:

The solution item y is undefined.

Response message string:

"The solution term y is undefined."

- `ok`

Corresponding constant:

`mosek.rescode.ok`

Description:

No error occurred.

Response message string:

"No error occurred."

- `trm_internal`

Corresponding constant:

`mosek.rescode.trm_internal`

Description:

The optimizer terminated due to some internal reason. Please contact MOSEK support.

Response message string:

"The optimizer terminated due to some internal reason."

- `trm_internal_stop`

Corresponding constant:

`mosek.rescode.trm_internal_stop`

Description:

The optimizer terminated for internal reasons. Please contact MOSEK support.

Response message string:

"The optimizer terminated for internal reasons."

- `trm_max_iterations`

Corresponding constant:

`mosek.rescode.trm_max_iterations`

Description:

The optimizer terminated at the maximum number of iterations.

Response message string:

"Maximum number of iterations is exceeded."

- `trm_max_num_setbacks`

Corresponding constant:

`mosek.rescode.trm_max_num_setbacks`

Description:

The optimizer terminated as the maximum number of set-backs was reached. This indicates numerical problems and a possibly badly formulated problem.

Response message string:

“The optimizer terminated as the maximum number of set-backs was reached.”

- `trm_max_time`

Corresponding constant:

`mosek.rescode.trm_max_time`

Description:

The optimizer terminated at the maximum amount of time.

Response message string:

“Maximum amount of time exceeded.”

- `trm_mio_near_abs_gap`

Corresponding constant:

`mosek.rescode.trm_mio_near_abs_gap`

Description:

The mixed-integer optimizer terminated because the near optimal absolute gap tolerance was satisfied.

Response message string:

“The mixed-integer optimizer terminated because the near optimal absolute gap tolerance was satisfied.”

- `trm_mio_near_rel_gap`

Corresponding constant:

`mosek.rescode.trm_mio_near_rel_gap`

Description:

The mixed-integer optimizer terminated because the near optimal relative gap tolerance was satisfied.

Response message string:

“The mixed-integer optimizer terminated because the near optimal relative gap tolerance was satisfied.”

- `trm_mio_num_branches`

Corresponding constant:

`mosek.rescode.trm_mio_num_branches`

Description:

The mixed-integer optimizer terminated as to the maximum number of branches was reached.

Response message string:

“The mixed-integer optimizer terminated as to the maximum number of branches was reached.”

- `trm_mio_num_relaxs`

Corresponding constant:

`mosek.rescode.trm_mio_num_relaxs`

Description:

The mixed-integer optimizer terminated as the maximum number of relaxations was reached.

Response message string:

“The mixed-integer optimizer terminated as the maximum number of relaxations was reached.”

- `trm_num_max_num_int_solutions`

Corresponding constant:

`mosek.rescode.trm_num_max_num_int_solutions`

Description:

The mixed-integer optimizer terminated as the maximum number of feasible solutions was reached.

Response message string:

“The mixed-integer optimizer terminated as the maximum number of feasible solutions was reached.”

- `trm_numerical_problem`

Corresponding constant:

`mosek.rescode.trm_numerical_problem`

Description:

The optimizer terminated due to numerical problems.

Response message string:

“The optimizer terminated due to a numerical problem.”

- `trm_objective_range`

Corresponding constant:

`mosek.rescode.trm_objective_range`

Description:

The optimizer terminated on the bound of the objective range.

Response message string:

“The optimal solution has an objective value outside the objective range.”

- `trm_stall`

Corresponding constant:

`mosek.rescode.trm_stall`

Description:

The optimizer terminated due to slow progress. Normally there are three possible reasons for this: Either a bug in MOSEK, the problem is badly formulated, or, in case of nonlinear problems, the nonlinear call-back functions are incorrect.

Please contact MOSEK support if this happens.

Response message string:

“The optimizer terminated due to slow progress.”

- `trm_user_break`

Corresponding constant:

`mosek.rescode.trm_user_break`

Description:

The optimizer terminated due to a user break.

Response message string:

“Control break was pressed.”

- `trm_user_callback`

Corresponding constant:

`mosek.rescode.trm_user_callback`

Description:

The optimizer terminated due to the return of the user-defined call-back function.

Response message string:

“The user-defined progress call-back function terminated the optimization.”

- `wrn_dropped_nz_qobj`

Corresponding constant:

`mosek.rescode.wrn_dropped_nz_qobj`

Description:

One or more non-zero elements were dropped in the Q matrix in the objective.

Response message string:

“’%d’ non-zero element(s) are dropped in the Q matrix in the objective.”

- `wrn_eliminator_space`

Corresponding constant:

`mosek.rescode.wrn_eliminator_space`

Description:

The eliminator is skipped at least once due to lack of space.

Response message string:

“The eliminator is skipped at least once due to lack of space.”

- `wrn_empty_name`

Corresponding constant:

`mosek.rescode.wrn_empty_name`

Description:

A variable or constraint name is empty. The output file may be invalid.

Response message string:

“A variable or constraint name is empty. The output file may be invalid.”

- `wrn_fixed_bound_values`

Corresponding constant:

`mosek.rescode.wrn_fixed_bound_values`

Description:

A fixed constraint/variable has been specified using the bound keys but the numerical bounds are different. The variable is fixed at the lower bound.

Response message string:

“For the bound key MSK_BK_FX the specified lower %24.16e and upper bound %24.16e are different.”

- `wrn_ignore_integer`

Corresponding constant:

`mosek.rescode.wrn_ignore_integer`

Description:

Ignored integer constraints.

Response message string:

“Ignored integer constraints.”

- `wrn_large_aij`

Corresponding constant:

`mosek.rescode.wrn_large_aij`

Description:

A numerically large value is specified for one $a_{i,j}$.

Response message string:

“A large value of %8.1e has been specified in A for variable '%s' (%d) in constraint '%s' (%d).”

- `wrn_large_bound`

Corresponding constant:

`mosek.rescode.wrn_large_bound`

Description:

A very large bound in absolute value has been specified.

Response message string:

“A large bound of value %8.1e has been specified for %s '%s' (%d).”

- `wrn_large_cj`

Corresponding constant:

`mosek.rescode.wrn_large_cj`

Description:

A numerically large value is specified for one c_j .

Response message string:

“A large value of %8.1e has been specified in cx for variable '%s' (%d).”

- `wrn_large_lo_bound`

Corresponding constant:

`mosek.rescode.wrn_large_lo_bound`

Description:

A large but finite lower bound in absolute value has been specified.

Response message string:

“A large lower bound of value %8.1e has been specified for %s '%s' (%d).”

- `wrn_large_up_bound`

Corresponding constant:

`mosek.rescode.wrn_large_up_bound`

Description:

A large but finite upper bound in absolute value has been specified.

Response message string:

“A large upper bound of value %8.1e has been specified for %s '%s' (%d).”

- `wrn_license_expire`

Corresponding constant:

`mosek.rescode.wrn_license_expire`

Description:

The license expires.

Response message string:

“The license expires in %ld days.”

- `wrn_license_feature_expire`

Corresponding constant:

`mosek.rescode.wrn_license_feature_expire`

Description:

The license expires.

Response message string:

“The license feature '%s' expires in %ld days.”

- `wrn_license_server`

Corresponding constant:

`mosek.rescode.wrn_license_server`

Description:

The license server is not responding.

Response message string:

“The license server is not responding.”

- `wrn_lp_drop_variable`

Corresponding constant:

`mosek.rescode.wrn_lp_drop_variable`

Description:

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.

Response message string:

“The variable '%s' is ignored because the variable was not previously defined.”

- `wrn_lp_old_quad_format`

Corresponding constant:

`mosek.rescode.wrn_lp_old_quad_format`

Description:

Missing `'/2'` after quadratic expressions in bound or objective.

Response message string:

“Missing `'/2'` after quadratic expressions in bound or objective.”

- `wrn_mio_infeasible_final`

Corresponding constant:

`mosek.rescode.wrn_mio_infeasible_final`

Description:

The final mixed integer problem with all the integer variables fixed at their optimal values is infeasible.

Response message string:

“The '%s' solution reports that final problem with all the integer variables fixed is infeasible while an integer solution has been found.”

- `wrn_mps_split_bou_vector`

Corresponding constant:

`mosek.rescode.wrn_mps_split_bou_vector`

Description:

A BOUNDS vector is split into several nonadjacent parts in an MPS file.

Response message string:

“The BOUNDS vector '%s' is split into several nonadjacent parts.”

- `wrn_mps_split_ran_vector`

Corresponding constant:

`mosek.rescode.wrn_mps_split_ran_vector`

Description:

A RANGE vector is split into several nonadjacent parts in an MPS file.

Response message string:

“The RANGE vector '%s' is split into several nonadjacent parts.”

- `wrn_mps_split_rhs_vector`

Corresponding constant:

`mosek.rescode.wrn_mps_split_rhs_vector`

Description:

An RHS vector is split into several nonadjacent parts in an MPS file.

Response message string:

“The RHS vector '%s' is split into several nonadjacent parts.”

- `wrn_name_max_len`

Corresponding constant:

`mosek.rescode.wrn_name_max_len`

Description:

A name is longer than the buffer that is supposed to hold it.

Response message string:

“A name of length %d is longer than the buffer of length %d that is supposed to hold it.”

- `wrn_no_global_optimizer`

Corresponding constant:

`mosek.rescode.wrn_no_global_optimizer`

Description:

No global optimizer is available.

Response message string:

“No global optimizer is available (%s).”

- `wrn_noncomplete_linear_dependency_check`

Corresponding constant:

`mosek.rescode.wrn_noncomplete_linear_dependency_check`

Description:

The linear dependency check(s) was not completed and therefore the A matrix may contain linear dependencies.

Response message string:

“The linear dependency check(s) is incomplete.”

- `wrn_nz_in_upr_tri`

Corresponding constant:

`mosek.rescode.wrn_nz_in_upr_tri`

Description:

Non-zero elements specified in the upper triangle of a matrix were ignored.

Response message string:

“Non-zero elements in the upper triangle of variable '%s' are ignored.”

- `wrn_open_param_file`

Corresponding constant:

`mosek.rescode.wrn_open_param_file`

Description:

The parameter file could not be opened.

Response message string:

“Could not open the parameter file '%s’.”

- `wrn_presolve_bad_precision`

Corresponding constant:

`mosek.rescode.wrn_presolve_bad_precision`

Description:

The presolve estimates that the model is specified with insufficient precision.

Response message string:

“The presolve estimates that the model is specified with insufficient precision.”

- `wrn_presolve_outofspace`

Corresponding constant:

`mosek.rescode.wrn_presolve_outofspace`

Description:

The presolve is incomplete due to lack of space.

Response message string:

“The presolve is incomplete due to lack of space.”

- `wrn_sol_filter`

Corresponding constant:

`mosek.rescode.wrn_sol_filter`

Description:

Invalid solution filter is specified.

Response message string:

“‘%s’ is an invalid solution filter is specified.”

- `wrn_spar_max_len`

Corresponding constant:

`mosek.rescode.wrn_spar_max_len`

Description:

A value for a string parameter is longer than the buffer that is supposed to hold it.

Response message string:

“A value for a string parameter is longer than the buffer that is supposed to hold it.”

- `wrn_too_few_basis_vars`

Corresponding constant:

`mosek.rescode.wrn_too_few_basis_vars`

Description:

An incomplete basis has been specified. Too few basis variables are specified.

Response message string:

“%d number of basis variables are specified but %d are expected.”

- `wrn_too_many_basis_vars`

Corresponding constant:

`mosek.rescode.wrn_too_many_basis_vars`

Description:

A basis with too many variables has been specified.

Response message string:

“%d basis variables are specified but %d are expected.”

- `wrn_undef_sol_file_name`

Corresponding constant:

`mosek.rescode.wrn_undef_sol_file_name`

Description:

Undefined name occurred in a solution.

Response message string:

“‘%s’ is an undefined %s name.”

- `wrn_using_generic_names`

Corresponding constant:

`mosek.rescode.wrn_using_generic_names`

Description:

The file writer reverts to generic names because a name is blank.

Response message string:

“The file writer reverts to generic names because a name is blank.”

- `wrn_write_discarded_cfix`

Corresponding constant:

`mosek.rescode.wrn_write_discarded_cfix`

Description:

The fixed objective term could not be converted to a variable and was discarded in the output file.

Response message string:

“The fixed objective term was discarded in the output file.”

- `wrn_zero_aij`

Corresponding constant:

`mosek.rescode.wrn.zero_aij`

Description:

One or more zero elements are specified in A.

Response message string:

“%d zero element(s) in A are specified.”

- `wrn.zeros_in_sparse_data`

Corresponding constant:

`mosek.rescode.wrn.zeros_in_sparse_data`

Description:

One or more almost zero elements are specified in sparse input data.

Response message string:

“%d zero elements are specified in sparse input data.”

Chapter 16

Constants

accmode	426
Constraint or variable access modes	
basindtype	426
Basis identification	
boundkey	426
Bound keys	
branchdir	427
Specifies the branching direction.	
callbackcode	427
Progress call-back codes	
checkconvexitytype	432
Types of convexity checks.	
compresstype	432
Compression types	
conetype	433
Cone types	
cputype	433
CPU type	
dataformat	434
Data format types	
dinfitem	434
Double information items	
dvalue	438
Double values	

<code>feasrepairtype</code>	438
Feasibility repair types	
<code>iinfitem</code>	438
Integer information items.	
<code>inftype</code>	443
Information item types	
<code>iomode</code>	443
Input/output modes	
<code>mark</code>	443
Bound keys	
<code>miocontsoltype</code>	443
Continuous mixed integer solution type	
<code>miomode</code>	444
Integer restrictions	
<code>mionodeseltype</code>	444
Mixed integer node selection types	
<code>mpsformattype</code>	444
MPS file format type	
<code>msgkey</code>	445
Message keys	
<code>networkdetect</code>	445
Network detection method	
<code>objsense</code>	445
Objective sense types	
<code>onoffkey</code>	445
On/off	
<code>optimizertype</code>	445
Optimizer types	
<code>orderingtype</code>	446
Ordering strategies	
<code>parametertype</code>	447
Parameter type	
<code>presolvemode</code>	447
Presolve method.	
<code>problemitem</code>	447
Problem data items	

problemtype	447
Problem types	
prosta	448
Problem status keys	
qreadtype	449
Interpretation of quadratic terms in MPS files	
rescodetype	449
Response code type	
scalingtype	449
Scaling type	
sensitivitytype	450
Sensitivity types	
simdegen	450
Degeneracy strategies	
simhotstart	450
Hot-start type employed by the simplex optimizer	
simseltype	450
Simplex selection strategy	
solitem	451
Solution items	
solsta	451
Solution status keys	
soltype	452
Solution types	
solveform	453
Solve primal or dual form	
stakey	453
Status keys	
startpointtype	453
Starting point types	
streamtype	454
Stream types	
value	454
Integer values	
variabletype	454
Variable types	

<code>xmlwriteroutputtype</code>	454
XML writer output mode	

16.1 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)

16.2 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.

16.3 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 a finite upper bound.

16.4 Specifies the branching direction.

- (2) `mosek.branchdir.down`
The mixed integer optimizer always chooses the down branch first.
- (0) `mosek.branchdir.free`
The mixed optimizer decides which branch to choose.
- (1) `mosek.branchdir.up`
The mixed integer optimizer always chooses the up branch first.

16.5 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_infeas_ana`
The call-back function is called when the infeasibility analyzer is started.
- (8) `mosek.callbackcode.begin_intpnt`
The call-back function is called when the interior-point optimizer is started.
- (9) `mosek.callbackcode.begin_license_wait`
Begin waiting for license.
- (10) `mosek.callbackcode.begin_mio`
The call-back function is called when the mixed integer optimizer is started.
- (11) `mosek.callbackcode.begin_network_dual_simplex`
The call-back function is called when the dual network simplex optimizer is started.

- (12) `mosek.callbackcode.begin_network_primal_simplex`
The call-back function is called when the primal network simplex optimizer is started.
- (13) `mosek.callbackcode.begin_network_simplex`
The call-back function is called when the simplex network optimizer is started.
- (14) `mosek.callbackcode.begin_nonconvex`
The call-back function is called when the nonconvex optimizer is started.
- (15) `mosek.callbackcode.begin_presolve`
The call-back function is called when the presolve is started.
- (16) `mosek.callbackcode.begin_primal_bi`
The call-back function is called from within the basis identification procedure when the primal phase is started.
- (17) `mosek.callbackcode.begin_primal_sensitivity`
Primal sensitivity analysis is started.
- (18) `mosek.callbackcode.begin_primal_setup_bi`
The call-back function is called when the primal BI setup is started.
- (19) `mosek.callbackcode.begin_primal_simplex`
The call-back function is called when the primal simplex optimizer is started.
- (20) `mosek.callbackcode.begin_simplex`
The call-back function is called when the simplex optimizer is started.
- (21) `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.
- (22) `mosek.callbackcode.begin_simplex_network_detect`
The call-back function is called when the network detection procedure is started.
- (23) `mosek.callbackcode.conic`
The call-back function is called from within the conic optimizer after the information database has been updated.
- (24) `mosek.callbackcode.dual_simplex`
The call-back function is called from within the dual simplex optimizer.
- (25) `mosek.callbackcode.end_bi`
The call-back function is called when the basis identification procedure is terminated.
- (26) `mosek.callbackcode.end_concurrent`
Concurrent optimizer is terminated.
- (27) `mosek.callbackcode.end_conic`
The call-back function is called when the conic optimizer is terminated.

- (28) `mosek.callbackcode.end_dual_bi`
The call-back function is called from within the basis identification procedure when the dual phase is terminated.
- (29) `mosek.callbackcode.end_dual_sensitivity`
Dual sensitivity analysis is terminated.
- (30) `mosek.callbackcode.end_dual_setup_bi`
The call-back function is called when the dual BI phase is terminated.
- (31) `mosek.callbackcode.end_dual_simplex`
The call-back function is called when the dual simplex optimizer is terminated.
- (32) `mosek.callbackcode.end_infeas_ana`
The call-back function is called when the infeasibility analyzer is terminated.
- (33) `mosek.callbackcode.end_intpnt`
The call-back function is called when the interior-point optimizer is terminated.
- (34) `mosek.callbackcode.end_license_wait`
End waiting for license.
- (35) `mosek.callbackcode.end_mio`
The call-back function is called when the mixed integer optimizer is terminated.
- (36) `mosek.callbackcode.end_network_dual_simplex`
The call-back function is called when the dual network simplex optimizer is terminated.
- (37) `mosek.callbackcode.end_network_primal_simplex`
The call-back function is called when the primal network simplex optimizer is terminated.
- (38) `mosek.callbackcode.end_network_simplex`
The call-back function is called when the simplex network optimizer is terminated.
- (39) `mosek.callbackcode.end_nonconvex`
The call-back function is called when the nonconvex optimizer is terminated.
- (40) `mosek.callbackcode.end_presolve`
The call-back function is called when the presolve is completed.
- (41) `mosek.callbackcode.end_primal_bi`
The call-back function is called from within the basis identification procedure when the primal phase is terminated.
- (42) `mosek.callbackcode.end_primal_sensitivity`
Primal sensitivity analysis is terminated.
- (43) `mosek.callbackcode.end_primal_setup_bi`
The call-back function is called when the primal BI setup is terminated.
- (44) `mosek.callbackcode.end_primal_simplex`
The call-back function is called when the primal simplex optimizer is terminated.

- (45) `mosek.callbackcode.end_simplex`
The call-back function is called when the simplex optimizer is terminated.
- (46) `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.
- (47) `mosek.callbackcode.end_simplex_network_detect`
The call-back function is called when the network detection procedure is terminated.
- (48) `mosek.callbackcode.ignore_value`
This code means that the call-back does not indicate a new phase in the optimization, but is simply a time-triggered call-back.
- (49) `mosek.callbackcode.im_bi`
The call-back function is called from within the basis identification procedure at an intermediate point.
- (50) `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.
- (51) `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.
- (52) `mosek.callbackcode.im_dual_sensitivity`
The call-back function is called at an intermediate stage of the dual sensitivity analysis.
- (53) `mosek.callbackcode.im_dual_simplex`
The call-back function is called at an intermediate point in the dual simplex optimizer.
- (54) `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.
- (55) `mosek.callbackcode.im_license_wait`
MOSEK is waiting for a license.
- (56) `mosek.callbackcode.im_mio`
The call-back function is called at an intermediate point in the mixed integer optimizer.
- (57) `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.
- (58) `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.

- (59) `mosek.callbackcode.im_mio_presolve`
The call-back function is called at an intermediate point in the mixed integer optimizer while running the presolve.
- (60) `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.
- (61) `mosek.callbackcode.im_network_dual_simplex`
The call-back function is called at an intermediate point in the dual network simplex optimizer.
- (62) `mosek.callbackcode.im_network_primal_simplex`
The call-back function is called at an intermediate point in the primal network simplex optimizer.
- (63) `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.
- (64) `mosek.callbackcode.im_presolve`
The call-back function is called from within the presolve procedure at an intermediate stage.
- (65) `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.
- (66) `mosek.callbackcode.im_primal_sensitivity`
The call-back function is called at an intermediate stage of the primal sensitivity analysis.
- (67) `mosek.callbackcode.im_primal_simplex`
The call-back function is called at an intermediate point in the primal simplex optimizer.
- (68) `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.
- (69) `mosek.callbackcode.intpnt`
The call-back function is called from within the interior-point optimizer after the information database has been updated.
- (70) `mosek.callbackcode.new_int_mio`
The call-back function is called after a new integer solution has been located by the mixed integer optimizer.
- (71) `mosek.callbackcode.noncover`
The call-back function is called from within the nonconvex optimizer after the information database has been updated.
- (72) `mosek.callbackcode.primal_simplex`
The call-back function is called from within the primal simplex optimizer.

- (73) `mosek.callbackcode.qcone`
The call-back function is called from within the Qcone optimizer.
- (74) `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.
- (75) `mosek.callbackcode.update_dual_simplex`
The call-back function is called in the dual simplex optimizer.
- (76) `mosek.callbackcode.update_network_dual_simplex`
The call-back function is called in the dual network simplex optimizer.
- (77) `mosek.callbackcode.update_network_primal_simplex`
The call-back function is called in the primal network simplex optimizer.
- (78) `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.
- (79) `mosek.callbackcode.update_presolve`
The call-back function is called from within the presolve procedure.
- (80) `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.
- (81) `mosek.callbackcode.update_primal_simplex`
The call-back function is called in the primal simplex optimizer.
- (82) `mosek.callbackcode.update_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.

16.6 Types of convexity checks.

- (0) `mosek.checkconvexitytype.none`
No convexity check.
- (1) `mosek.checkconvexitytype.simple`
Perform simple and fast convexity check.

16.7 Compression types

- (1) `mosek.compressstype.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.

16.8 Cone types

- (0) `mosek.conetype.quad`
The cone is a quadratic cone.
- (1) `mosek.conetype.rquad`
The cone is a rotated quadratic cone.

16.9 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.

16.10 Data format types

- (0) `mosek.dataformat.extension`
The file extension is used to determine the data file format.
- (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.

16.11 Double information items

- (0) `mosek.dinfitem.bi_clean_cputime`
Time (in CPU seconds) spent within the clean-up phase of the basis identification procedure since its invocation.
- (1) `mosek.dinfitem.bi_cputime`
Time (in CPU seconds) spent within the basis identification procedure since its invocation.
- (2) `mosek.dinfitem.bi_dual_cputime`
Time (in CPU seconds) spent within the dual phase basis identification procedure since its invocation.
- (3) `mosek.dinfitem.bi_primal_cputime`
Time (in CPU seconds) spent within the primal phase of the basis identification procedure since its invocation.
- (4) `mosek.dinfitem.concurrent_cputime`
Time (in CPU seconds) spent within the concurrent optimizer since its invocation.
- (5) `mosek.dinfitem.concurrent_realtime`
Time (in wall-clock seconds) within the concurrent optimizer since its invocation.
- (6) `mosek.dinfitem.intpnt_cputime`
Time (in CPU seconds) spent within the interior-point optimizer since its invocation.
- (7) `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.)

- (8) `mosek.dinfitem.intpnt_dual_obj`
Dual objective value reported by the interior-point or Qcone optimizer.
- (9) `mosek.dinfitem.intpnt_factor_num_flops`
An estimate of the number of flops used in the factorization.
- (10) `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.
- (11) `mosek.dinfitem.intpnt_order_cputime`
Order time (in CPU seconds).
- (12) `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).
- (13) `mosek.dinfitem.intpnt_primal_obj`
Primal objective value reported by the interior-point or Qcone optimizer.
- (14) `mosek.dinfitem.intpnt_realtime`
Time (in wall-clock seconds) spent within the interior-point optimizer since its invocation.
- (15) `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.
- (16) `mosek.dinfitem.mio_cputime`
Time spent in the mixed integer optimizer.
- (17) `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.

- (18) `mosek.dinfitem.mio_obj_bound`
The best bound objective value corresponding to the best integer feasible solution is located. Please note that at least one integer feasible solution must be located i.e. check `mosek.iinfitem.mio_num_int_solutions`.
- (19) `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`.

- (20) `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

$$\frac{|(\text{objective value of feasible solution}) - (\text{objective bound})|}{\max(1, |(\text{objective value of feasible solution})|)}.$$

Otherwise it has the value -1.0.

- (21) `mosek.dinfitem.mio_user_obj_cut`
 If the objective cut is used, then this information item has the value of the cut.
- (22) `mosek.dinfitem.optimizer_cputime`
 Total time (in CPU seconds) spent in the optimizer since it was invoked.
- (23) `mosek.dinfitem.optimizer_realttime`
 Total time (in wall-clock seconds) spent in the optimizer since it was invoked.
- (24) `mosek.dinfitem.presolve_cputime`
 Total time (in CPU seconds) spent in the presolve since it was invoked.
- (25) `mosek.dinfitem.presolve_eli_cputime`
 Total time (in CPU seconds) spent in the eliminator since the presolve was invoked.
- (26) `mosek.dinfitem.presolve_lindep_cputime`
 Total time (in CPU seconds) spent in the linear dependency checker since the presolve was invoked.
- (27) `mosek.dinfitem.rd_cputime`
 Time (in CPU seconds) spent reading the data file.
- (28) `mosek.dinfitem.sim_cputime`
 Time (in CPU seconds) spent in the simplex optimizer since invoking it.
- (29) `mosek.dinfitem.sim_feas`
 Feasibility measure reported by the simplex optimizer.
- (30) `mosek.dinfitem.sim_obj`
 Objective value reported by the simplex optimizer.
- (31) `mosek.dinfitem.sol_bas_dual_obj`
 Dual objective value of the basic solution. Updated at the end of the optimization.
- (32) `mosek.dinfitem.sol_bas_max_dbi`
 Maximal dual bound infeasibility in the basic solution. Updated at the end of the optimization.
- (33) `mosek.dinfitem.sol_bas_max_deqi`
 Maximal dual equality infeasibility in the basic solution. Updated at the end of the optimization.
- (34) `mosek.dinfitem.sol_bas_max_pbi`
 Maximal primal bound infeasibility in the basic solution. Updated at the end of the optimization.

- (35) `mosek.dinfitem.sol_bas_max_peqi`
Maximal primal equality infeasibility in the basic solution. Updated at the end of the optimization.
- (36) `mosek.dinfitem.sol_bas_max_pinti`
Maximal primal integer infeasibility in the basic solution. Updated at the end of the optimization.
- (37) `mosek.dinfitem.sol_bas_primal_obj`
Primal objective value of the basic solution. Updated at the end of the optimization.
- (38) `mosek.dinfitem.sol_int_max_pbi`
Maximal primal bound infeasibility in the integer solution. Updated at the end of the optimization.
- (39) `mosek.dinfitem.sol_int_max_peqi`
Maximal primal equality infeasibility in the basic solution. Updated at the end of the optimization.
- (40) `mosek.dinfitem.sol_int_max_pinti`
Maximal primal integer infeasibility in the integer solution. Updated at the end of the optimization.
- (41) `mosek.dinfitem.sol_int_primal_obj`
Primal objective value of the integer solution. Updated at the end of the optimization.
- (42) `mosek.dinfitem.sol_itr_dual_obj`
Dual objective value of the interior-point solution. Updated at the end of the optimization.
- (43) `mosek.dinfitem.sol_itr_max_dbi`
Maximal dual bound infeasibility in the interior-point solution. Updated at the end of the optimization.
- (44) `mosek.dinfitem.sol_itr_max_dcni`
Maximal dual cone infeasibility in the interior-point solution. Updated at the end of the optimization.
- (45) `mosek.dinfitem.sol_itr_max_deqi`
Maximal dual equality infeasibility in the interior-point solution. Updated at the end of the optimization.
- (46) `mosek.dinfitem.sol_itr_max_pbi`
Maximal primal bound infeasibility in the interior-point solution. Updated at the end of the optimization.
- (47) `mosek.dinfitem.sol_itr_max_pcni`
Maximal primal cone infeasibility in the interior-point solution. Updated at the end of the optimization.
- (48) `mosek.dinfitem.sol_itr_max_peqi`
Maximal primal equality infeasibility in the interior-point solution. Updated at the end of the optimization.

- (49) `mosek.dinfitem.sol_itr_max_pinti`
Maximal primal integer infeasibility in the interior-point solution. Updated at the end of the optimization.
- (50) `mosek.dinfitem.sol_itr_primal_obj`
Primal objective value of the interior-point solution. Updated at the end of the optimization.

16.12 Double values

- (1e+30) `mosek.dvalue.infinity`
Definition of infinity.

16.13 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.

16.14 Integer information items.

- (0) `mosek.iinfitem.bi_iter`
Number of simplex pivots performed since invoking the basis identification procedure.
- (1) `mosek.iinfitem.cache_size_l1`
L1 cache size used.
- (2) `mosek.iinfitem.cache_size_l2`
L2 cache size used.
- (3) `mosek.iinfitem.concurrent_fastest_optimizer`
The type of the optimizer that finished first in a concurrent optimization.
- (4) `mosek.iinfitem.cpu_type`
The type of cpu detected.
- (5) `mosek.iinfitem.intpnt_factor_num_nz`
Number of non-zeros in factorization.
- (6) `mosek.iinfitem.intpnt_factor_num_offcol`
Number of columns in the constraint matrix (or Jacobian) that has an offending structure.

- (7) `mosek.iinfitem.intpnt_iter`
Number of interior-point iterations since invoking the interior-point optimizer.
- (8) `mosek.iinfitem.intpnt_num_threads`
Number of threads that the interior-point optimizer is using.
- (9) `mosek.iinfitem.intpnt_solve_dual`
Non-zero if the interior-point optimizer is solving the dual problem.
- (10) `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.
- (11) `mosek.iinfitem.mio_initial_solution`
Is non-zero if an initial integer solution is specified.
- (12) `mosek.iinfitem.mio_num_active_nodes`
Number of active nodes in the branch and bound tree.
- (13) `mosek.iinfitem.mio_num_branch`
Number of branches performed during the optimization.
- (14) `mosek.iinfitem.mio_num_cuts`
Number of cuts generated by the mixed integer optimizer.
- (15) `mosek.iinfitem.mio_num_int_solutions`
Number of integer feasible solutions that has been found.
- (16) `mosek.iinfitem.mio_num_intpnt_iter`
Number of interior-point iterations performed by the mixed-integer optimizer.
- (17) `mosek.iinfitem.mio_num_relax`
Number of relaxations solved during the optimization.
- (18) `mosek.iinfitem.mio_num_simplex_iter`
Number of simplex iterations performed by the mixed-integer optimizer.
- (19) `mosek.iinfitem.mio_numcon`
Number of constraints in the problem solved by the mixed integer optimizer.
- (20) `mosek.iinfitem.mio_numint`
Number of integer variables in the problem solved by the mixed integer optimizer.
- (21) `mosek.iinfitem.mio_numvar`
Number of variables in the problem solved by the mixed integer optimizer.
- (22) `mosek.iinfitem.mio_total_num_basis_cuts`
Number of basis cuts.
- (23) `mosek.iinfitem.mio_total_num_branch`
Number of branches performed during the optimization.

- (24) `mosek.iinfitem.mio_total_num_cardgub_cuts`
Number of cardgub cuts.
- (25) `mosek.iinfitem.mio_total_num_clique_cuts`
Number of clique cuts.
- (26) `mosek.iinfitem.mio_total_num_coef_redc_cuts`
Number of coef. redc. cuts.
- (27) `mosek.iinfitem.mio_total_num_contra_cuts`
Number of contra cuts.
- (28) `mosek.iinfitem.mio_total_num_cuts`
Total number of cuts generated by the mixed integer optimizer.
- (29) `mosek.iinfitem.mio_total_num_disagg_cuts`
Number of diasagg cuts.
- (30) `mosek.iinfitem.mio_total_num_flow_cover_cuts`
Number of flow cover cuts.
- (31) `mosek.iinfitem.mio_total_num_gcd_cuts`
Number of gcd cuts.
- (32) `mosek.iinfitem.mio_total_num_gomory_cuts`
Number of Gomory cuts.
- (33) `mosek.iinfitem.mio_total_num_gub_cover_cuts`
Number of GUB cover cuts.
- (34) `mosek.iinfitem.mio_total_num_knapsur_cover_cuts`
Number of knapsack cover cuts.
- (35) `mosek.iinfitem.mio_total_num_lattice_cuts`
Number of lattice cuts.
- (36) `mosek.iinfitem.mio_total_num_lift_cuts`
Number of lift cuts.
- (37) `mosek.iinfitem.mio_total_num_obj_cuts`
Number of obj cuts.
- (38) `mosek.iinfitem.mio_total_num_plan_loc_cuts`
Number of loc cuts.
- (39) `mosek.iinfitem.mio_total_num_relax`
Number of relaxations solved during the optimization.
- (40) `mosek.iinfitem.mio_user_obj_cut`
If it is non-zero, then the objective cut is used.
- (41) `mosek.iinfitem.opt_numcon`
Number of constraints in the problem solved when the optimizer is called.

- (42) `mosek.iinfitem.opt_numvar`
Number of variables in the problem solved when the optimizer is called
- (43) `mosek.iinfitem.optimize_response`
The response code returned by optimize.
- (44) `mosek.iinfitem.rd_numanz`
Number of non-zeros in A that is read.
- (45) `mosek.iinfitem.rd_numcon`
Number of constraints read.
- (46) `mosek.iinfitem.rd_numcone`
Number of conic constraints read.
- (47) `mosek.iinfitem.rd_numintvar`
Number of integer constrained variables read.
- (48) `mosek.iinfitem.rd_numq`
Number of nonempty Q matrices read.
- (49) `mosek.iinfitem.rd_numqnz`
Number of Q non-zeros.
- (50) `mosek.iinfitem.rd_numvar`
Number of variables read.
- (51) `mosek.iinfitem.rd_prototype`
Problem type.
- (52) `mosek.iinfitem.sim_dual_deg_iter`
The number of dual degenerate iterations.
- (53) `mosek.iinfitem.sim_dual_hotstart`
If 1 then the dual simplex algorithm is solving from an advance basis.
- (54) `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.
- (55) `mosek.iinfitem.sim_dual_inf_iter`
The number of iterations taken with dual infeasibility.
- (56) `mosek.iinfitem.sim_dual_iter`
Number of dual simplex iterations during the last optimization.
- (57) `mosek.iinfitem.sim_numcon`
Number of constraints in the problem solved by the simplex optimizer.
- (58) `mosek.iinfitem.sim_numvar`
Number of variables in the problem solved by the simplex optimizer.

- (59) `mosek.iinfitem.sim_primal_deg_iter`
The number of primal degenerate iterations.
- (60) `mosek.iinfitem.sim_primal_hotstart`
If 1 then the primal simplex algorithm is solving from an advance basis.
- (61) `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.
- (62) `mosek.iinfitem.sim_primal_inf_iter`
The number of iterations taken with primal infeasibility.
- (63) `mosek.iinfitem.sim_primal_iter`
Number of primal simplex iterations during the last optimization.
- (64) `mosek.iinfitem.sim_solve_dual`
Is non-zero if dual problem is solved.
- (65) `mosek.iinfitem.sol_bas_prosta`
Problem status of the basic solution. Updated after each optimization.
- (66) `mosek.iinfitem.sol_bas_solsta`
Solution status of the basic solution. Updated after each optimization.
- (67) `mosek.iinfitem.sol_int_prosta`
Problem status of the integer solution. Updated after each optimization.
- (68) `mosek.iinfitem.sol_int_solsta`
Solution status of the integer solution. Updated after each optimization.
- (69) `mosek.iinfitem.sol_itr_prosta`
Problem status of the interior-point solution. Updated after each optimization.
- (70) `mosek.iinfitem.sol_itr_solsta`
Solution status of the interior-point solution. Updated after each optimization.
- (71) `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.
- (72) `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.
- (73) `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 .

16.15 Information item types

- (0) `mosek.inftype.dou_type`
Is a double information type.
- (1) `mosek.inftype.int_type`
Is an integer.

16.16 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.

16.17 Bound keys

- (0) `mosek.mark.lo`
The lower bound is selected for sensitivity analysis.
- (1) `mosek.mark.up`
The upper bound is selected for sensitivity analysis.

16.18 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.

16.19 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.

16.20 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.

16.21 MPS file format type

- (2) `mosek.mpsformattype.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.mpsformattype.relaxed`
It is assumed that the input file satisfies a slightly relaxed version of the MPS format.
- (0) `mosek.mpsformattype.strict`
It is assumed that the input file satisfies the MPS format strictly.

16.22 Message keys

(1100) `mosek.msgkey.mps_selected`

(1000) `mosek.msgkey.reading_file`

(1001) `mosek.msgkey.writing_file`

16.23 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.

16.24 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.

16.25 On/off

(0) `mosek.onoffkey.off`
Switch the option off.

(1) `mosek.onoffkey.on`
Switch the option on.

16.26 Optimizer types

(9) `mosek.optimizertype.concurrent`
The optimizer for nonconvex nonlinear problems.

- (2) `mosek.optimizertype.conic`
Another cone optimizer.
- (5) `mosek.optimizertype.dual_simplex`
The dual simplex optimizer is used.
- (0) `mosek.optimizertype.free`
The optimizer is chosen automatically.
- (6) `mosek.optimizertype.free_simplex`
Either the primal or the dual simplex optimizer is used.
- (1) `mosek.optimizertype.intpnt`
The interior-point optimizer is used.
- (7) `mosek.optimizertype.mixed_int`
The mixed integer optimizer.
- (8) `mosek.optimizertype.nonconvex`
The optimizer for nonconvex nonlinear problems.
- (4) `mosek.optimizertype.primal_simplex`
The primal simplex optimizer is used.
- (3) `mosek.optimizertype.qcone`
The Qcone optimizer is used.

16.27 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.

16.28 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.

16.29 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.

16.30 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.

16.31 Problem types

- (4) `mosek.problemtype.conic`
A conic optimization.
- (3) `mosek.problemtype.geco`
General convex optimization.

- (0) `mosek.problemtype.lo`
The problem is a linear optimization problem.
- (5) `mosek.problemtype.mixed`
General nonlinear constraints and conic constraints. This combination can not be solved by MOSEK.
- (2) `mosek.problemtype.qcqp`
The problem is a quadratically constrained optimization problem.
- (1) `mosek.problemtype.qo`
The problem is a quadratic optimization problem.

16.32 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.

16.33 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.

16.34 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.

16.35 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.

16.36 Sensitivity types

- (0) `mosek.sensitivitytype.basis`
Basis sensitivity analysis is performed.
- (1) `mosek.sensitivitytype.optimal_partition`
Optimal partition sensitivity analysis is performed.

16.37 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.

16.38 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.

16.39 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).

16.40 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.

16.41 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.

16.42 Solution types

- (1) `mosek.soltype.bas`
The basic solution.
- (2) `mosek.soltype.itg`
The integer solution.
- (0) `mosek.soltype.itr`
The interior solution.

16.43 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.

16.44 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.

16.45 Starting point types

- (1) `mosek.startpointtype.constant`
The starting point is set to a constant. This is more reliable than a non-constant starting point.
- (0) `mosek.startpointtype.free`
The starting point is chosen automatically.

16.46 Stream types

- (2) `mosek.streamtype.err`
Error stream.
- (0) `mosek.streamtype.log`
Log stream.
- (1) `mosek.streamtype.msg`
Message stream.
- (3) `mosek.streamtype.wrn`
Warning stream.

16.47 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.

16.48 Variable types

- (0) `mosek.variabletype.type_cont`
Is a continuous variable.
- (1) `mosek.variabletype.type_int`
Is an integer variable.

16.49 XML writer output mode

- (1) `mosek.xmlwriteroutputtype.col`
Write in column order.
- (0) `mosek.xmlwriteroutputtype.row`
Write in row order.

Appendix A

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.

A.1 The MPS file format

The version of the MPS format supported by MOSEK allows specification of an optimization problem on the form

$$\begin{aligned} l^c &\leq Ax + q(x) &\leq u^c, \\ l^x &\leq x &\leq u^x, \\ x &\in \mathcal{C}, \\ x_{\mathcal{J}} &\text{ integer}, \end{aligned} \tag{A.1}$$

where

- $x \in R^n$ is the vector of decision variables.
- $A \in R^{m \times n}$ is the constraint matrix.
- $l^c \in R^m$ is the lower limit on the activity for the constraints.
- $u^c \in R^m$ is the upper limit on the activity for the constraints.
- $l^x \in R^n$ is the lower limit on the activity for the variables.
- $u^x \in R^n$ is the upper limit on the activity for the variables.
- $q : R^n \rightarrow R$ is a vector of quadratic functions. Hence,

$$q_i(x) = 1/2 x^T Q^i x$$

where it is assumed that

$$Q^i = (Q^i)^T. \tag{A.2}$$

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 A.5 for details.

A.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.

A.1.2 NAME

In this section a name ([name]) is assigned to the problem.

A.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.

A.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.

A.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 .

A.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.

A.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.

A.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			

A.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 .

A.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].

A.1.11 CSECTION (optional)

The purpose of the CSECTION is to specify the constraint

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

in (A.1).

It is assumed that \mathcal{C} satisfies the following requirements. Let

$$x^t \in R^{n^t}, \quad t = 1, \dots, k$$

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

$$\mathcal{C} := \{x \in R^n : x^t \in \mathcal{C}_t, \quad t = 1, \dots, k\}$$

where \mathcal{C}_t must have one of the following forms

- R set:

$$\mathcal{C}_t = \{x \in R^{n^t}\}.$$

- Quadratic cone:

$$\mathcal{C}_t = \left\{ x \in R^{n^t} : x_1 \geq \sqrt{\sum_{j=2}^{n^t} x_j^2} \right\}. \quad (\text{A.3})$$

- Rotated quadratic cone:

$$\mathcal{C}_t = \left\{ x \in R^{n^t} : 2x_1x_2 \geq \sum_{j=3}^{n^t} x_j^2, \quad x_1, x_2 \geq 0 \right\}. \quad (\text{A.4})$$

In general, only quadratic and rotated quadratic cones are specified in the MPS file whereas membership of the 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 R cone.

Next, let us study an example. Assume that the quadratic cone

$$x_4 \geq \sqrt{x_5^2 + x_8^2} \quad (\text{A.5})$$

and the rotated quadratic cone

$$2x_3x_7 \geq x_1^2 + x_2^2, \quad x_3, x_7 \geq 0, \quad (\text{A.6})$$

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 (A.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 (A.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. (A.3).
RQUAD	≥ 2	Rotated quadratic cone i.e. (A.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`.

A.1.12 ENDATA

This keyword denotes the end of the MPS file.

A.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.

A.3 General limitations

- An MPS file should be an ASCII file.

A.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.

A.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 B

The LP file format

MOSEK supports the LP file format with some extensions i.e. MOSEK can read and write LP formatted files.

B.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.

B.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

- $x \in R^n$ is the vector of decision variables.
- $c \in R^n$ is the linear term in the objective.
- $q^o : R^n \rightarrow R$ is the quadratic term in the objective where

$$q^o(x) = x^T Q^o x$$

and it is assumed that

$$Q^o = (Q^o)^T. \tag{B.1}$$

- $A \in R^{m \times n}$ is the constraint matrix.
- $l^c \in R^m$ is the lower limit on the activity for the constraints.
- $u^c \in R^m$ is the upper limit on the activity for the constraints.
- $l^x \in R^n$ is the lower limit on the activity for the variables.
- $u^x \in R^n$ is the upper limit on the activity for the variables.
- $q : R^n \rightarrow R$ is a vector of quadratic functions. Hence,

$$q_i(x) = x^T Q^i x$$

where it is assumed that

$$Q^i = (Q^i)^T. \quad (\text{B.2})$$

- $\mathcal{J} \subseteq \{1, 2, \dots, n\}$ is an index set of the integer constrained variables.

B.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.

B.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 $4 \text{ } x1 + 2 \text{ } x1$ is equivalent to $6 \text{ } x1$. In the quadratic expressions $x1 * x2$ is equivalent to $x2 * x1$ and as in the linear part, if the same variables multiplied or squared occur several times their coefficients are added.

B.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.

$$-5 \leq x_1 + x_2 \leq 5 \tag{B.3}$$

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 (B.3) may be written as

$$x_1 + x_2 - sl_1 = 0, \quad -5 \leq sl_1 \leq 5.$$

B.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
```

B.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.

B.2.1.5 Terminating section

Finally, an LP formatted file must be terminated with the keyword

```
end
```

B.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
```

B.2.2 LP format peculiarities

B.2.2.1 Comments

Anything on a line after a “\” is ignored and is treated as a comment.

B.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.

B.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.

B.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 satisfy 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.

B.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.

B.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.

B.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

B.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 C

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.

C.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.

C.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 + y + x^2 + 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]
```

C.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

$$-10 \leq x, y \leq 10. \quad (\text{C.1})$$

- **[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 thing is always to set to status 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**.

- **[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.

C.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]+)?
```

C.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"
```


C.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]
```

C.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.

C.5 Examples

This section contains a set of small examples written in OPF and describing how to formulate linear, quadratic and conic problems.

C.5.1 Linear example lo1.opf

Consider the example:

$$\begin{aligned}
 &\text{minimize} && -10x_1 && -9x_2, \\
 &\text{subject to} && 7/10x_1 + && 1x_2 &\leq 630, \\
 & && 1/2x_1 + && 5/6x_2 &\leq 600, \\
 & && 1x_1 + && 2/3x_2 &\leq 708, \\
 & && 1/10x_1 + && 1/4x_2 &\leq 135, \\
 & && x_1, && x_2 &\geq 0.
 \end{aligned} \tag{C.2}$$

In the OPF format the example is displayed as shown below:

```
[comment]
  Example lo1.mps converted to OPF.
[/comment]

[hints]
  # Give a hint about the size of the different elements in the problem.
  # These need only be estimates, but in this case they are exact.
  [hint NUMVAR] 2 [/hint]
  [hint NUMCON] 4 [/hint]
  [hint NUMANZ] 8 [/hint]
[/hints]

[variables]
  # All variables that will appear in the problem
  x1 x2
[/variables]

[objective minimize 'obj']
  - 10 x1 - 9 x2
[/objective]

[constraints]
  [con 'c1'] 0.7 x1 +          x2 <= 630 [/con]
  [con 'c2'] 0.5 x1 + 0.8333333333 x2 <= 600 [/con]
  [con 'c3']      x1 + 0.666666667 x2 <= 708 [/con]
  [con 'c4'] 0.1 x1 + 0.25      x2 <= 135 [/con]
[/constraints]

[bounds]
  # By default all variables are free. The following line will
  # change this to all variables being nonnegative.
  [b] 0 <= * [/b]
[/bounds]
```

C.5.2 Quadratic example qo1.opf

An example of a quadratic optimization problem is

$$\begin{aligned} & \text{minimize} && x_1^2 + 0.1x_2^2 + x_3^2 - x_1x_3 - x_2 \\ & \text{subject to} && 1 \leq x_1 + x_2 + x_3, \\ & && x \geq 0. \end{aligned} \tag{C.3}$$

This can be formulated in `opf` as shown below.

```
[comment]
  Example qo1.mps converted to OPF.
[/comment]

[hints]
  [hint NUMVAR] 3 [/hint]
  [hint NUMCON] 1 [/hint]
  [hint NUMANZ] 3 [/hint]
[/hints]

[variables]
  x1 x2 x3
[/variables]

[objective minimize 'obj']
  # The quadratic terms are often multiplied by 1/2,
  # but this is not required.

  - x2 + 0.5 ( 2 x1 ^ 2 - 2 x3 * x1 + 0.2 x2 ^ 2 + 2 x3 ^ 2 )
[/objective]

[constraints]
  [con 'c1'] 1 <= x1 + x2 + x3 [/con]
[/constraints]

[bounds]
  [b] 0 <= * [/b]
[/bounds]
```

C.5.3 Conic quadratic example cqo1.opf

Consider the example:

$$\begin{aligned} & \text{minimize} && 1x_1 + 2x_2 \\ & \text{subject to} && 2x_3 + 4x_4 = 5, \\ & && x_5^2 \leq 2x_1x_3, \\ & && x_6^2 \leq 2x_2x_4, \\ & && x_5 = 1, \\ & && x_6 = 1, \\ & && x \geq 0. \end{aligned} \tag{C.4}$$

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.

```

[comment]
  Example cq01.mps converted to OPF.
[/comment]

[hints]
  [hint NUMVAR] 6 [/hint]
  [hint NUMCON] 1 [/hint]
  [hint NUMANZ] 2 [/hint]
[/hints]

[variables]
  x1 x2 x3 x4 x5 x6
[/variables]

[objective minimize 'obj']
  x1 + 2 x2
[/objective]

[constraints]
  [con 'c1'] 2 x3 + 4 x4 = 5 [/con]
[/constraints]

[bounds]
  # We let all variables default to the positive orthant
  [b] 0 <= * [/b]
  # ... and change those that differ from the default.
  [b] x5,x6 = 1 [/b]

  # We define two rotated quadratic cones

  # k1: 2 x1 * x3 >= x5^2
  [cone rquad 'k1'] x1, x3, x5 [/cone]

  # k2: 2 x2 * x4 >= x6^2
  [cone rquad 'k2'] x2, x4, x6 [/cone]
[/bounds]

```

C.5.4 Mixed integer example milo1.opf

Consider the mixed integer problem:

$$\begin{aligned}
 & \text{maximize} && x_0 + 0.64x_1 \\
 & \text{subject to} && 50x_0 + 31x_1 \leq 250, \\
 & && 3x_0 - 2x_1 \geq -4, \\
 & && x_0, x_1 \geq 0 \quad \text{and integer}
 \end{aligned} \tag{C.5}$$

This can be implemented in OPF with:

```

[comment]
  Written by MOSEK version 5.0.0.7
  Date 20-11-106
  Time 14:42:24
[/comment]

```

```
[hints]
  [hint NUMVAR] 2 [/hint]
  [hint NUMCON] 2 [/hint]
  [hint NUMANZ] 4 [/hint]
[/hints]

[variables disallow_new_variables]
  x1 x2
[/variables]

[objective maximize 'obj']
  x1 + 6.4e-1 x2
[/objective]

[constraints]
  [con 'c1']          5e+1 x1 + 3.1e+1 x2 <= 2.5e+2 [/con]
  [con 'c2'] -4 <= 3 x1 - 2 x2 [/con]
[/constraints]

[bounds]
  [b] 0 <= * [/b]
[/bounds]

[integer]
  x1 x2
[/integer]
```


Appendix D

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 E

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 E.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 E.1: The standard ORD format.

E.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 F

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.

F.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 F.1: Status keys.

NAME The name of the constraint assigned by the user.

AT The status of the constraint. In Table F.1 the possible values of the status keys and their interpretation are shown.

ACTIVITY Given the i th constraint on the form

$$l_i^c \leq \sum_{j=1}^n a_{ij}x_j \leq u_i^c, \quad (\text{F.1})$$

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 (F.1)).

UPPER LIMIT Is the quantity u_i^c (see (F.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.

F.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] K. D. Andersen. A Modified Schur Complement Method for Handling Dense Columns in Interior-Point Methods for Linear Programming. *ACM Trans. Math. Software*, 22(3):348–356, 1996.
- [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. Kennington 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, 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] R. J. Vanderbei. *Linear Programming. Foundations and Extensions*. Kluwer Academic Publishers, Boston/London/Dordrecht, 1997.
- [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, 70
- mosek.accmode**, 426
- `alloc_add_qnz` (parameter), 276
- append
 - `Task.append` (example), 23
- `append` (Task method), 158
- `appendcone`
 - `Task.appendcone` (example), 33
- `appendcone` (Task method), 158
- `appendcons` (Task method), 159
- `appendstat` (Task method), 160
- `appendvars` (Task method), 160
- attaching streams, 17

- `bas_sol_file_name` (parameter), 335
- mosek.basindtype**, 426
- basis identification, 78
- `basis_rel_tol_s` (parameter), 248
- `basis_tol_s` (parameter), 249
- `basis_tol_x` (parameter), 249
- `bi_clean_optimizer` (parameter), 276
- `bi_ignore_max_iter` (parameter), 276
- `bi_ignore_num_error` (parameter), 277
- `bi_lu_tol_rel_piv` (parameter), 249
- `bi_max_iterations` (parameter), 277
- mosek.boundkey**, 426
- bounds, infinite, 56
- mosek.branchdir**, 427

- `cache_size_l1` (parameter), 277
- `cache_size_l2` (parameter), 278
- `callback_freq` (parameter), 249
- mosek.callbackcode**, 427
- certificate
 - dual, 58
 - primal, 57
- `check_convexity` (parameter), 278
- `check_ctrl_c` (parameter), 278
- `check_task_data` (parameter), 279
- `checkconvexity` (Task method), 161
- mosek.checkconvexitytype**, 432
- `checkdata` (Task method), 161
- `checkmemtask` (Task method), 161
- `chgbound` (Task method), 161
- `clonetask` (Task method), 162
- `commitchanges` (Task method), 162
- complementarity conditions, 57
- mosek.compresstype**, 432
- con
 - `mosek.accmode.con` (example), 23
- concurrent
 - `mosek.optimizertype.concurrent` (example), 84
- concurrent optimization, 83
- concurrent solution, 83
- concurrent-num-optimizers
 - `mosek.iparam.concurrent_num_optimizers` (example), 84
- `concurrent_num_optimizers` (parameter), 279
- `concurrent_priority_dual_simplex` (parameter), 279
- `concurrent_priority_free_simplex` (parameter), 279
- `concurrent_priority_intpnt` (parameter), 280
- `concurrent_priority_primal_simplex` (parameter), 280
- mosek.conetype**, 433
- conic, 32
 - optimization, 61
 - problem, 61
- conic modelling, 63
 - minimizing norms, example, 64
 - pitfalls, 67
 - quadratic objective, example, 63
 - risk and market impact, example
 - Markowitz model, example, 71

- conic optimization, 32
- conic problem example, 33
- conic quadratic optimization, 32
- constraint
 - matrix, 55, 455
 - quadratic, 59
- constraints
 - lower limit, 56, 455
 - number of, 16
 - upper limit, 56, 455
- continuous relaxation, 89
- convex quadratic problem, 25
- cpu.type (parameter), 280
- mosek.cputype**, 433
-
- data.check (parameter), 281
- data.file.name (parameter), 335
- data.tol.aij (parameter), 250
- data.tol.aij_large (parameter), 250
- data.tol.bound_inf (parameter), 250
- data.tol.bound_wrn (parameter), 250
- data.tol.c.huge (parameter), 251
- data.tol.cj_large (parameter), 251
- data.tol.qij (parameter), 251
- data.tol.x (parameter), 251
- mosek.dataformat**, 434
- debug.file.name (parameter), 336
- deletesolution (Task method), 162
- mosek.dinfitem**, 434
- dual certificate, 58
- dual infeasible, 56, 58
- duality gap (linear problem), 57
- dualizer, 77
- dualsensitivity
 - Task.dualsensitivity (example), 118
- dualsensitivity (Task method), 163
- mosek.dvalue**, 438
-
- echointro (Env method), 145
- eliminator, 76
- Embedded network flow problems, 81
- Env
 - constructors, 143
- env, creating, 14
- env, initializing, 14
- Env.Env (constructor), 143
- err_api_array_too_small (response code), 361
- err_api_callback (response code), 361
- err_api_cb_connect (response code), 362
- err_api_fatal_error (response code), 362
- err_api_internal (response code), 362
- err_api_nl_data (response code), 362
- err_argument_dimension (response code), 362
- err_argument_lenneq (response code), 362
- err_argument_perm_array (response code), 363
- err_argument_type (response code), 363
- err_basis (response code), 363
- err_basis_factor (response code), 363
- err_basis_singular (response code), 363
- err_cannot_clone_nl (response code), 363
- err_cannot_handle_nl (response code), 364
- err_con_q_not_nsd (response code), 364
- err_con_q_not_psd (response code), 364
- err_concurrent_optimizer (response code), 364
- err_cone_index (response code), 365
- err_cone_overlap (response code), 365
- err_cone_rep_var (response code), 365
- err_cone_size (response code), 365
- err_cone_type (response code), 365
- err_cone_type_str (response code), 365
- err_data_file_ext (response code), 366
- err_dup_name (response code), 366
- err_end_of_file (response code), 366
- err_factor (response code), 366
- err_feasrepair_cannot_relax (response code), 366
- err_feasrepair_inconsistent_bound (response code), 367
- err_feasrepair_solving_relaxed (response code), 367
- err_file_license (response code), 367
- err_file_open (response code), 367
- err_file_read (response code), 367
- err_file_write (response code), 368
- err_first (response code), 368
- err_firsti (response code), 368
- err_firstj (response code), 368
- err_flexlm (response code), 368
- err_huge_c (response code), 369
- err_identical_tasks (response code), 369
- err_in_argument (response code), 369
- err_index (response code), 369
- err_index_arr_is_too_large (response code), 369
- err_index_arr_is_too_small (response code), 370

- err_index_is_too_large (response code), 370
- err_index_is_too_small (response code), 370
- err_inf_dou_index (response code), 370
- err_inf_dou_name (response code), 370
- err_inf_int_index (response code), 371
- err_inf_int_name (response code), 371
- err_inf_type (response code), 371
- err_infinite_bound (response code), 371
- err_internal (response code), 371
- err_internal_test_failed (response code), 371
- err_inv_aptre (response code), 372
- err_inv_bk (response code), 372
- err_inv_bkc (response code), 372
- err_inv_bkx (response code), 372
- err_inv_cone_type (response code), 372
- err_inv_cone_type_str (response code), 373
- err_inv_marki (response code), 373
- err_inv_markj (response code), 373
- err_inv_name_item (response code), 373
- err_inv_numi (response code), 373
- err_inv_numj (response code), 374
- err_inv_optimizer (response code), 374
- err_inv_problem (response code), 374
- err_inv_qcon_subi (response code), 374
- err_inv_qcon_subj (response code), 374
- err_inv_qcon_subk (response code), 375
- err_inv_qcon_val (response code), 375
- err_inv_qobj_subi (response code), 375
- err_inv_qobj_subj (response code), 375
- err_inv_qobj_val (response code), 375
- err_inv_sk (response code), 376
- err_inv_sk_str (response code), 376
- err_inv_skc (response code), 376
- err_inv_skn (response code), 376
- err_inv_skx (response code), 376
- err_inv_var_type (response code), 377
- err_invalid_accmode (response code), 377
- err_invalid_ampl_stub (response code), 377
- err_invalid_branch_direction (response code), 377
- err_invalid_branch_priority (response code), 377
- err_invalid_compression (response code), 377
- err_invalid_file_name (response code), 378
- err_invalid_format_type (response code), 378
- err_invalid_iomode (response code), 378
- err_invalid_mbt_file (response code), 378
- err_invalid_name_in_sol_file (response code), 378
- err_invalid_obj_name (response code), 379
- err_invalid_objective_sense (response code), 379
- err_invalid_sol_file_name (response code), 379
- err_invalid_stream (response code), 379
- err_invalid_task (response code), 379
- err_invalid_utf8 (response code), 380
- err_invalid_wchar (response code), 380
- err_last (response code), 380
- err_lasti (response code), 380
- err_lastj (response code), 380
- err_license (response code), 381
- err_license_cannot_allocate (response code), 381
- err_license_cannot_connect (response code), 381
- err_license_expired (response code), 381
- err_license_feature (response code), 381
- err_license_invalid_hostid (response code), 382
- err_license_max (response code), 382
- err_license_moseklm_daemon (response code), 382
- err_license_server (response code), 382
- err_license_server_version (response code), 382
- err_license_version (response code), 383
- err_link_file_dll (response code), 383
- err_lp_dup_slack_name (response code), 383
- err_lp_empty (response code), 383
- err_lp_file_format (response code), 383
- err_lp_format (response code), 383
- err_lp_free_constraint (response code), 384
- err_lp_incompatible (response code), 384
- err_lp_invalid_var_name (response code), 384
- err_lp_write_conic_problem (response code), 384
- err_lp_write_geco_problem (response code), 384
- err_lu_max_num_tries (response code), 385
- err_maxnumanz (response code), 385
- err_maxnumcon (response code), 385
- err_maxnumcone (response code), 385
- err_maxnumqnz (response code), 386
- err_maxnumvar (response code), 386
- err_mbt_incompatible (response code), 386
- err_mio_no_optimizer (response code), 386
- err_mio_not_loaded (response code), 386
- err_missing_license_file (response code), 387
- err_mixed_problem (response code), 387
- err_mps_cone_overlap (response code), 387
- err_mps_cone_repeat (response code), 387

- `err_mps_cone_type` (response code), 387
- `err_mps_file` (response code), 388
- `err_mps_inv_bound_key` (response code), 388
- `err_mps_inv_con_key` (response code), 388
- `err_mps_inv_field` (response code), 388
- `err_mps_inv_marker` (response code), 388
- `err_mps_inv_sec_name` (response code), 389
- `err_mps_inv_sec_order` (response code), 389
- `err_mps_invalid_obj_name` (response code), 389
- `err_mps_invalid_objsense` (response code), 389
- `err_mps_mul_con_name` (response code), 389
- `err_mps_mul_csec` (response code), 390
- `err_mps_mul_qobj` (response code), 390
- `err_mps_mul_qsec` (response code), 390
- `err_mps_no_objective` (response code), 390
- `err_mps_null_con_name` (response code), 390
- `err_mps_null_var_name` (response code), 390
- `err_mps splitted_var` (response code), 391
- `err_mps_tab_in_field2` (response code), 391
- `err_mps_tab_in_field3` (response code), 391
- `err_mps_tab_in_field5` (response code), 391
- `err_mps_undef_con_name` (response code), 391
- `err_mps_undef_var_name` (response code), 392
- `err_mula_element` (response code), 392
- `err_name_max_len` (response code), 392
- `err_nan_in_blc` (response code), 392
- `err_nan_in_blx` (response code), 392
- `err_nan_in_buc` (response code), 393
- `err_nan_in_bux` (response code), 393
- `err_nan_in_c` (response code), 393
- `err_nan_in_double_data` (response code), 393
- `err_negative_append` (response code), 393
- `err_negative_surplus` (response code), 394
- `err_newer_dll` (response code), 394
- `err_no_basis_sol` (response code), 394
- `err_no_dual_for_itg_sol` (response code), 394
- `err_no_dual_infeas_cer` (response code), 394
- `err_no_init_env` (response code), 395
- `err_no_optimizer_var_type` (response code), 395
- `err_no_primal_infeas_cer` (response code), 395
- `err_no_solution_in_callback` (response code), 395
- `err_nonconvex` (response code), 395
- `err_nonlinear_equality` (response code), 395
- `err_nonlinear_ranged` (response code), 396
- `err_nr_arguments` (response code), 396
- `err_null_env` (response code), 396
- `err_null_name` (response code), 396
- `err_null_pointer` (response code), 396
- `err_null_task` (response code), 397
- `err_numconlim` (response code), 397
- `err_numvarlim` (response code), 397
- `err_obj_q_not_nsd` (response code), 397
- `err_obj_q_not_psd` (response code), 397
- `err_objective_range` (response code), 398
- `err_older_dll` (response code), 398
- `err_open_dl` (response code), 398
- `err_opf_format` (response code), 398
- `err_optimizer_license` (response code), 399
- `err_ord_invalid` (response code), 399
- `err_ord_invalid_branch_dir` (response code), 399
- `err_param_index` (response code), 399
- `err_param_is_too_large` (response code), 399
- `err_param_is_too_small` (response code), 399
- `err_param_name` (response code), 400
- `err_param_name_dou` (response code), 400
- `err_param_name_int` (response code), 400
- `err_param_name_str` (response code), 400
- `err_param_type` (response code), 400
- `err_param_value_str` (response code), 401
- `err_platform_not_licensed` (response code), 401
- `err_postsolve` (response code), 401
- `err_pro_item` (response code), 401
- `err_prob_license` (response code), 401
- `err_qcon_subi_too_large` (response code), 402
- `err_qcon_subi_too_small` (response code), 402
- `err_qcon_upper_triangle` (response code), 402
- `err_qobj_upper_triangle` (response code), 402
- `err_read_format` (response code), 402
- `err_readlp_nonexisting_name` (response code), 403
- `err_remove_cone_variable` (response code), 403
- `err_sen_bound_invalid_lo` (response code), 403
- `err_sen_bound_invalid_up` (response code), 403
- `err_sen_format` (response code), 403
- `err_sen_index_invalid` (response code), 404
- `err_sen_index_range` (response code), 404
- `err_sen_invalid_regexp` (response code), 404
- `err_sen_numerical` (response code), 404
- `err_sen_solution_status` (response code), 404
- `err_sen_undef_name` (response code), 405
- `err_size_license` (response code), 405
- `err_size_license_con` (response code), 405

- `err.size_license_intvar` (response code), 405
- `err.size_license_var` (response code), 405
- `err.sol_file_number` (response code), 406
- `err.solitem` (response code), 406
- `err.solver_probtype` (response code), 406
- `err.space` (response code), 406
- `err.space_leaking` (response code), 406
- `err.space_no_info` (response code), 407
- `err.thread_cond_init` (response code), 407
- `err.thread_create` (response code), 407
- `err.thread_mutex_init` (response code), 407
- `err.thread_mutex_lock` (response code), 407
- `err.thread_mutex_unlock` (response code), 408
- `err.too_small_maxnumanz` (response code), 408
- `err.unb_step_size` (response code), 408
- `err.undef_solution` (response code), 408
- `err.undefined_objective_sense` (response code), 408
- `err.unknown` (response code), 409
- `err.user_func_ret` (response code), 409
- `err.user_func_ret_data` (response code), 409
- `err.user_nlo_eval` (response code), 409
- `err.user_nlo_eval_hessubi` (response code), 409
- `err.user_nlo_eval_hessubj` (response code), 410
- `err.user_nlo_func` (response code), 410
- `err.whichitem_not_allowed` (response code), 410
- `err.whichsol` (response code), 410
- `err.write_lp_format` (response code), 410
- `err.write_lp_non_unique_name` (response code), 411
- `err.write_mps_invalid_name` (response code), 411
- `err.write_opf_invalid_var_name` (response code), 411
- `err.xml_invalid_problem_type` (response code), 411
- `err.y_is_undefined` (response code), 411
- Error
 - constructors, 148
- Error.Exception (constructor), 149
- example
 - conic problem, 33
 - `cqo1.py`, 33
 - `lo1.py`, 17
 - `lo1`, 17
 - `lo2`, 22
 - `lo2.python`, 23
 - `milol.py`, 36
 - `miointsol.py`, 38
 - `qo1.py`, 26
 - `qo1`, 26
 - quadratic constraints, 29
 - quadratic objective, 26
 - simple, 14
- Exception
 - constructors, 149
- Exception.Exception (constructor), 149
- executing Python script (Linux), 11
- feasible, primal, 56
- feasrepair-optimize
 - `mosek.iparam.feasrepair_optimize` (example), 108
- `feasrepair_name_prefix` (parameter), 336
- `feasrepair_name_separator` (parameter), 336
- `feasrepair_name_wsumviol` (parameter), 336
- `feasrepair_optimize` (parameter), 281
- `feasrepair_tol` (parameter), 252
- `mosek.feasrepairtype`, 438
- `flush_stream_freq` (parameter), 281
- `getaij` (Task method), 163
- `getapieceenumnz` (Task method), 164
- `getaslice` (Task method), 164
- `getaslicenumnz` (Task method), 165
- `getaslicetrip` (Task method), 165
- `getavec` (Task method), 166
- `getavecnumnz` (Task method), 166
- `getbound` (Task method), 167
- `getboundslice` (Task method), 167
- `getc` (Task method), 168
- `getcfix` (Task method), 168
- `getcone` (Task method), 168
- `getconeinfo` (Task method), 168
- `getconname` (Task method), 169
- `getcslice` (Task method), 169
- `getdouinf` (Task method), 169
- `getdoupam` (Task method), 170
- `getdualobj` (Task method), 170
- `getinfeasiblesubproblem` (Task method), 170
- `getintinf` (Task method), 170
- `getintparam` (Task method), 171
- `getmaxnamelen` (Task method), 171
- `getmaxnumanz` (Task method), 171

- getmaxnumcon (Task method), 171
- getmaxnumcone (Task method), 171
- getmaxnumqnz (Task method), 172
- getmaxnumvar (Task method), 172
- getmemusagetask (Task method), 172
- getname (Task method), 172
- getnameindex (Task method), 173
- getnumanz (Task method), 173
- getnumcon (Task method), 173
- getnumcone (Task method), 173
- getnumconemem (Task method), 174
- getnumintvar (Task method), 174
- getnumparam (Task method), 174
- getnumqconnz (Task method), 174
- getnumqobjnz (Task method), 175
- getnumvar (Task method), 175
- getobjname (Task method), 175
- getobjsense (Task method), 175
- getprimalobj
 - Task.getprimalobj (example), 108
- getprimalobj (Task method), 175
- getprobtype (Task method), 176
- getqconk (Task method), 176
- getqobj (Task method), 176
- getqobjij (Task method), 177
- getreducedcosts (Task method), 177
- getsolution (Task method), 177
- getsolutioni (Task method), 179
- getsolutioninf (Task method), 180
- getsolutionslice
 - Task.getsolutionslice (example), 17
- getsolutionslice (Task method), 180
- getsolutionstatus (Task method), 182
- getsolutionstatuskeyslice (Task method), 182
- getstrparam (Task method), 183
- getsymbcondim (Env method), 145
- gettaskname (Task method), 183
- getvarbranchdir (Task method), 183
- getvarbranchpri (Task method), 183
- getvarname (Task method), 184
- getvartype (Task method), 184
- getvartypelist (Task method), 184
- getversion (Env method), 145
- help desk, 7
- hot-start, 79
- mosek.iinfitem, 438
- infeas_generic_names (parameter), 282
- infeas_prefer_primal (parameter), 282
- infeas_report_auto (parameter), 282
- infeas_report_level (parameter), 282
- infeasible, 95
 - dual, 58
 - primal, 57
- infeasible problems, 95
- infeasible, dual, 56
- infeasible, primal, 56
- infinite bounds, 56
- mosek.inftype, 443
- initbasissolve (Task method), 184
- initenv
 - Env.initenv (example), 14, 17
- initenv (Env method), 145
- inputdata
 - Task.inputdata (example), 17, 29, 118
- inputdata (Task method), 185
- installation, 9
- int_sol_file_name (parameter), 337
- integer optimization, 35, 89
 - relaxation, 89
- interior-point optimizer, 78, 82
- interior-point or simplex optimizer, 80
- intpnt_basis (parameter), 283
- intpnt_co_tol_dfeas (parameter), 252
- intpnt_co_tol_infeas (parameter), 252
- intpnt_co_tol_mu_red (parameter), 252
- intpnt_co_tol_near_rel (parameter), 253
- intpnt_co_tol_pfeas (parameter), 253
- intpnt_co_tol_rel_gap (parameter), 253
- intpnt_diff_step (parameter), 283
- intpnt_factor_debug_lvl (parameter), 284
- intpnt_factor_method (parameter), 284
- intpnt_max_iterations (parameter), 284
- intpnt_max_num_cor (parameter), 284
- intpnt_max_num_refinement_steps (parameter), 284
- intpnt_nl_merit_bal (parameter), 253
- intpnt_nl_tol_dfeas (parameter), 254
- intpnt_nl_tol_mu_red (parameter), 254
- intpnt_nl_tol_near_rel (parameter), 254
- intpnt_nl_tol_pfeas (parameter), 255
- intpnt_nl_tol_rel_gap (parameter), 255

- `intpnt.nl.tol.rel.step` (parameter), 255
- `intpnt.num.threads` (parameter), 285
- `intpnt.off.col.trh` (parameter), 285
- `intpnt.order.method` (parameter), 285
- `intpnt.regularization.use` (parameter), 286
- `intpnt.scaling` (parameter), 286
- `intpnt.solve.form` (parameter), 286
- `intpnt.starting.point` (parameter), 287
- `intpnt.tol.dfeas` (parameter), 255
- `intpnt.tol.dsafes` (parameter), 255
- `intpnt.tol.infeas` (parameter), 256
- `intpnt.tol.mu.red` (parameter), 256
- `intpnt.tol.path` (parameter), 256
- `intpnt.tol.pfeas` (parameter), 257
- `intpnt.tol.psafe` (parameter), 257
- `intpnt.tol.rel.gap` (parameter), 257
- `intpnt.tol.rel.step` (parameter), 257
- `intpnt.tol.step.size` (parameter), 258
- `mosek.iomode`, 443
- `isdouppname` (Task method), 186
- `isinfinity` (Env method), 146
- `isintpname` (Task method), 186
- `isstrpname` (Task method), 186
- `itg`
 - `mosek.soltype.itg` (example), 36, 38
- `itr`
 - `mosek.soltype.itr` (example), 17, 29
- `itr_sol_file_name` (parameter), 337
- `license.allow.overuse` (parameter), 287
- `license.cache.time` (parameter), 287
- `license.check.time` (parameter), 288
- `license.debug` (parameter), 288
- `license.pause.time` (parameter), 288
- `license.suppress.expire.wrns` (parameter), 288
- `license.wait` (parameter), 289
- linear dependency check, 76
- Linear network flow problems, 59
- linear optimization, 16
- linear problem, 55
- linearity interval, 112
- `linkfiletoenvstream` (Env method), 146
- `linkfiletotaskstream` (Task method), 187
- `log`
 - `mosek.streamtype.log` (example), 17
- `log` (parameter), 289
- `log.bi` (parameter), 289
- `log.bi.freq` (parameter), 290
- `log.concurrent` (parameter), 290
- `log.cut.second.opt` (parameter), 290
- `log.factor` (parameter), 291
- `log.feasrepair` (parameter), 291
- `log.file` (parameter), 291
- `log.head` (parameter), 291
- `log.infeas.ana` (parameter), 292
- `log.intpnt` (parameter), 292
- `log.mio` (parameter), 292
- `log.mio.freq` (parameter), 292
- `log.nonconvex` (parameter), 293
- `log.optimizer` (parameter), 293
- `log.order` (parameter), 293
- `log.param` (parameter), 293
- `log.presolve` (parameter), 294
- `log.response` (parameter), 294
- `log.sensitivity` (parameter), 294
- `log.sensitivity.opt` (parameter), 295
- `log.sim` (parameter), 295
- `log.sim.freq` (parameter), 295
- `log.sim.minor` (parameter), 295
- `log.sim.network.freq` (parameter), 296
- `log.storage` (parameter), 296
- `lower.obj.cut` (parameter), 258
- `lower.obj.cut.finite.trh` (parameter), 258
- LP format, 467
- `lp.write.ignore.incompatible.items` (parameter), 296
- `makesolutionstatusunknown` (Task method), 187
- `mosek.mark`, 443
- matrix format
 - column ordered, 51
 - row ordered, 51
 - triplets, 51
- `max.num.warnings` (parameter), 296
- `maxnumanz.double.trh` (parameter), 297
- `mio.branch.dir` (parameter), 297
- `mio.branch.priorities.use` (parameter), 297
- `mio.construct.sol` (parameter), 297
- `mio.cont.sol` (parameter), 298
- `mio.cut.level.root` (parameter), 298
- `mio.cut.level.tree` (parameter), 299
- `mio.disable.term.time` (parameter), 258

- `mio.feaspump_level` (parameter), 299
- `mio.heuristic_level` (parameter), 299
- `mio.heuristic_time` (parameter), 259
- `mio.keep_basis` (parameter), 300
- `mio.local_branch_number` (parameter), 300
- `mio.max_num_branches` (parameter), 300
- `mio.max_num_relaxs` (parameter), 301
- `mio.max_num_solutions` (parameter), 301
- `mio.max_time` (parameter), 259
- `mio.max_time_aprx_opt` (parameter), 259
- `mio.mode` (parameter), 301
- `mio.near_tol_abs_gap` (parameter), 260
- `mio.near_tol_rel_gap` (parameter), 260
- `mio.node_optimizer` (parameter), 302
- `mio.node_selection` (parameter), 302
- `mio.presolve_aggregate` (parameter), 303
- `mio.presolve_probing` (parameter), 303
- `mio.presolve_use` (parameter), 303
- `mio.rel_add_cut_limited` (parameter), 260
- `mio.root_optimizer` (parameter), 303
- `mio.strong_branch` (parameter), 304
- `mio.tol_abs_gap` (parameter), 261
- `mio.tol_abs_relax_int` (parameter), 261
- `mio.tol_rel_gap` (parameter), 261
- `mio.tol_rel_relax_int` (parameter), 261
- `mio.tol_x` (parameter), 262
- `mosek.miocontsoltype`, 443
- `mosek.miomode`, 444
- `mosek.mionodeseltype`, 444
- mixed integer optimization, 35, 89
- modelling
 - absolute value, 70
 - in cones, 63
 - market impact term, 72
 - Markowitz portfolio optimization, 71
 - minimizing a sum of norms, 64
 - portfolio optimization, 71
 - transaction costs, 72
- MPS format, 455
 - BOUNDS, 462
 - COLUMNS, 458
 - free, 466
 - NAME, 457
 - OBJNAME, 458
 - OBJSENSE, 457
 - QSECTION, 460
 - RANGES, 460
 - RHS, 459
 - ROWS, 458
- `mosek.mpsformattype`, 444
- msg
 - `mosek.streamtype.msg` (example), 17
- `mosek.msgkey`, 445
- Network flow problems
 - embedded, 81
 - formulating, 59
 - optimizing, 81
- `mosek.networkdetect`, 445
- `nonconvex_max_iterations` (parameter), 304
- `nonconvex_tol_feas` (parameter), 262
- `nonconvex_tol_opt` (parameter), 262
- objective
 - defining, 17
 - linear, 17
 - quadratic, 59
 - vector, 55
- objective-sense
 - `mosek.iparam.objective_sense` (example), 17
- `objective_sense` (parameter), 304
- `mosek.objsense`, 445
- ok (response code), 412
- `mosek.onoffkey`, 445
- OPF format, 475
- OPF, writing, 14
- opf, writing
 - OPF, writing (example), 14
- opf-write-hints
 - `mosek.iparam.opf.write_hints` (example), 14
- opf-write-parameters
 - `mosek.iparam.opf.write_parameters` (example), 14
- opf-write-problem
 - `mosek.iparam.opf.write_problem` (example), 14
- opf-write-solutions
 - `mosek.iparam.opf.write_solutions` (example), 14
- `opf_max_terms_per_line` (parameter), 305
- `opf_write_header` (parameter), 305
- `opf_write_hints` (parameter), 305
- `opf_write_parameters` (parameter), 306
- `opf_write_problem` (parameter), 306
- `opf_write_sol_bas` (parameter), 306

- `opf.write_sol_itg` (parameter), 306
- `opf.write_sol_itr` (parameter), 307
- `opf.write_solutions` (parameter), 307
- optimal solution, 57
- optimization
 - conic, 61
 - integer, 35, 89
 - mixed integer, 35, 89
- optimize
 - `Task.optimize` (example), 14, 17, 118
- `optimize` (Task method), 187
- optimize-penalty
 - `mosek.feasrepairtype.optimize_penalty` (example), 108
- optimizeconcurrent
 - `Task.optimizeconcurrent` (example), 86
- `optimizeconcurrent` (Task method), 187
- optimizer
 - `mosek.iparam.optimizer` (example), 84
- `optimizer` (parameter), 307
- `optimizer_max_time` (parameter), 262
- optimizers
 - concurrent, 83
 - conic interior-point, 82
 - convex interior-point, 82
 - linear interior-point, 78
 - parallel, 83
 - simplex, 79
- `mosek.optimizertype`, 445
- `optimizetrm` (Task method), 188
- Optimizing
 - network flow problems, 81
- ORD format, 489
- `mosek.orderingtype`, 446
- parallel extensions, 83
- parallel interior-point, 77
- parallel optimizers
 - interior point, 77
- parallel solution, 83
- `param.comment_sign` (parameter), 337
- `param.read_case_name` (parameter), 308
- `param.read_file_name` (parameter), 337
- `param.read_ign_error` (parameter), 308
- `param.write_file_name` (parameter), 338
- `mosek.parametertype`, 447
- presolve, 75
 - eliminator, 76
 - linear dependency check, 76
- `presolve_elim_fill` (parameter), 308
- `presolve_eliminator_use` (parameter), 309
- `presolve_level` (parameter), 309
- `presolve_lindep_use` (parameter), 309
- `presolve_lindep_work_lim` (parameter), 309
- `presolve_tol_aij` (parameter), 263
- `presolve_tol_lin_dep` (parameter), 263
- `presolve_tol_s` (parameter), 263
- `presolve_tol_x` (parameter), 263
- `presolve_use` (parameter), 310
- `mosek.presolvemode`, 447
- primal feasible, 56
- primal certificate, 57
- primal infeasible, 56, 57
- primal-dual solution, 56
- primalsensitivity
 - `Task.primalsensitivity` (example), 118
- `primalsensitivity` (Task method), 188
- `printdata` (Task method), 190
- `printparam` (Task method), 191
- problem element
 - bounds
 - constraint, 17
 - variable, 17
 - constraint
 - bounds, 17
 - constraint matrix, 17
 - objective, linear, 17
 - variable
 - bounds, 17
 - variable vector, 16
- `mosek.problemitem`, 447
- `mosek.problemtyp`, 447
- `mosek.prosta`, 448
- `putaij` (Task method), 191
- `putaijlist` (Task method), 192
- putavec
 - `Task.putavec` (example), 23
- `putavec` (Task method), 192
- `putaveclist` (Task method), 193
- `putbound` (Task method), 194
- `putboundlist` (Task method), 195
- `putboundslice`

- Task.putboundslice (example), 23
- putboundslice (Task method), 195
- putcfix
 - Task.putcfix (example), 23
- putcfix (Task method), 196
- putcj (Task method), 196
- putclist
 - Task.putclist (example), 23
- putclist (Task method), 196
- putcone (Task method), 197
- putcpudefaults (Env method), 146
- putdllpath (Env method), 146
- putdoupparam (Task method), 197
- putintparam
 - Task.putintparam (example), 17, 84
- putintparam (Task method), 197
- putkeepdlls (Env method), 147
- putlicensedefaults (Env method), 147
- putmaxnumanz (Task method), 197
- putmaxnumcon (Task method), 198
- putmaxnumcone (Task method), 198
- putmaxnumqnz (Task method), 199
- putmaxnumvar (Task method), 199
- putnadoupparam (Task method), 199
- putnaintparam (Task method), 199
- putname (Task method), 200
- putnastrparam (Task method), 200
- putobjname (Task method), 200
- putobjsense (Task method), 200
- putparam (Task method), 201
- putqcon (Task method), 201
- putqconk
 - Task.putqconk (example), 29
- putqconk (Task method), 202
- putqobj
 - Task.putqobj (example), 26
- putqobj (Task method), 203
- putqobjij (Task method), 203
- putsolution (Task method), 204
- putsolutioni (Task method), 205
- putsolutionyi (Task method), 205
- putstrparam (Task method), 206
- puttaskname (Task method), 206
- putvarbranchorder (Task method), 206
- putvartype (Task method), 206
- putvartypelist
 - Task.putvartypelist (example), 36, 38
- putvartypelist (Task method), 207
- Python script
 - executing (Linux), 11
- Python on Linux, 10
- Python on Windows, 9
- mosek.qreadtype, 449
- quad
 - mosek.conetype.quad (example), 33
- quadratic constraint, 59
- quadratic constraints, example, 29
- quadratic objective, 59
- quadratic objective, example, 26
- quadratic optimization, 25, 59
- quadratic problem, 25
- read_add_anz (parameter), 310
- read_add_con (parameter), 310
- read_add_cone (parameter), 311
- read_add_qnz (parameter), 311
- read_add_var (parameter), 311
- read_anz (parameter), 311
- read_con (parameter), 311
- read_cone (parameter), 312
- read_data_compressed (parameter), 312
- read_data_format (parameter), 312
- read_keep_free_con (parameter), 313
- read_lp_drop_new_vars_in_bou (parameter), 313
- read_lp_quoted_names (parameter), 313
- read_mps_bou_name (parameter), 338
- read_mps_format (parameter), 314
- read_mps_keep_int (parameter), 314
- read_mps_obj_name (parameter), 338
- read_mps_obj_sense (parameter), 314
- read_mps_quoted_names (parameter), 314
- read_mps_ran_name (parameter), 338
- read_mps_relax (parameter), 315
- read_mps_rhs_name (parameter), 339
- read_mps_width (parameter), 315
- read_q_mode (parameter), 315
- read_qnz (parameter), 316
- read_task_ignore_param (parameter), 316
- read_var (parameter), 316
- readbranchpriorities (Task method), 207
- readdata
 - Task.readdata (example), 14, 108

- readdata (Task method), 207
- readparamfile (Task method), 208
- readsolution (Task method), 208
- readsummary (Task method), 208
- relaxation, continuous, 89
- relaxprimal
 - Task.relaxprimal (example), 108
- relaxprimal (Task method), 208
- remove (Task method), 211
- removecone (Task method), 211
- mosek.rescodetype**, 449
- resizetask (Task method), 211
- rquad
 - mosek.conetype.rquad (example), 33
- scaling, 77
- mosek.scalingtype**, 449
- sensitivity analysis, 111
 - basis type, 113
 - optimal partition type, 114
- sensitivity_all (parameter), 316
- sensitivity_file_name (parameter), 339
- sensitivity_optimizer (parameter), 317
- sensitivity_res_file_name (parameter), 339
- sensitivity_type (parameter), 317
- sensitivityreport (Task method), 212
- mosek.sensitivitytype**, 450
- set_Stream (Env method), 147
- set_Stream (Task method), 212
- setdefaults (Task method), 212
- shadow price, 112
- sim_degen (parameter), 318
- sim_dual_crash (parameter), 318
- sim_dual_restrict_selection (parameter), 318
- sim_dual_selection (parameter), 319
- sim_hotstart (parameter), 319
- sim_max_iterations (parameter), 319
- sim_max_num_setbacks (parameter), 320
- sim_network_detect (parameter), 320
- sim_network_detect_hotstart (parameter), 320
- sim_network_detect_method (parameter), 321
- sim_non_singular (parameter), 321
- sim_primal_crash (parameter), 321
- sim_primal_restrict_selection (parameter), 322
- sim_primal_selection (parameter), 322
- sim_refactor_freq (parameter), 322
- sim_save_lu (parameter), 323
- sim_scaling (parameter), 323
- sim_solve_form (parameter), 323
- sim_stability_priority (parameter), 324
- sim_switch_optimizer (parameter), 324
- mosek.simdegen**, 450
- mosek.simhotstart**, 450
- simplex optimizer, 79
- simplex_abs_tol_piv (parameter), 264
- mosek.simseltype**, 450
- sol_filter_keep_basic (parameter), 324
- sol_filter_keep_ranged (parameter), 325
- sol_filter_xc_low (parameter), 339
- sol_filter_xc_upr (parameter), 340
- sol_filter_xx_low (parameter), 340
- sol_filter_xx_upr (parameter), 340
- sol_quoted_names (parameter), 325
- sol_read_name_width (parameter), 325
- sol_read_width (parameter), 325
- mosek.solitem**, 451
- mosek.solsta**, 451
- mosek.soltype**, 452
- solution, primal-dual, 56
- solution, optimal, 57
- solution_callback (parameter), 326
- solutiondef
 - Task.solutiondef (example), 36, 38
- solutiondef (Task method), 213
- solutionsummary
 - Task.solutionsummary (example), 14, 17, 86
- solutionsummary (Task method), 213
- mosek.solveform**, 453
- solvewithbasis (Task method), 213
- sparse vector, 50
- mosek.stakey**, 453
- mosek.startpointtype**, 453
- startstat (Task method), 214
- stat_file_name (parameter), 341
- stat_key (parameter), 341
- stat_name (parameter), 341
- stopstat (Task method), 214
- stream
 - attaching, 17
- mosek.streamtype**, 454
- strtoconetype (Task method), 215
- strtosk (Task method), 215

- Task
 - constructors, 149
 - task, creatig, 14
 - Task.Task (constructor), 149
 - thread safety, 126
 - trm_internal (response code), 412
 - trm_internal_stop (response code), 412
 - trm_max_iterations (response code), 412
 - trm_max_num_setbacks (response code), 412
 - trm_max_time (response code), 413
 - trm_mio_near_abs_gap (response code), 413
 - trm_mio_near_rel_gap (response code), 413
 - trm_mio_num_branches (response code), 413
 - trm_mio_num_relaxes (response code), 413
 - trm_num_max_num_int_solutions (response code), 414
 - trm_numerical_problem (response code), 414
 - trm_objective_range (response code), 414
 - trm_stall (response code), 414
 - trm_user_break (response code), 415
 - trm_user_callback (response code), 415
 - undefsolution (Task method), 215
 - upper_obj_cut (parameter), 264
 - upper_obj_cut_finite_trh (parameter), 264
 - mosek.value, 454
 - var
 - mosek.accmode.var (example), 23
 - variables
 - decision, 55, 455
 - lower limit, 56, 455
 - number of, 16
 - upper limit, 56, 455
 - mosek.variabletype, 454
 - vector format
 - full, 50
 - sparse, 50
- Warning
 - constructors, 217
 - Warning.Warning (constructor), 217
 - warning_level (parameter), 326
 - write-data-format
 - mosek.iparam.write_data_format (example), 14
 - write_bas_constraints (parameter), 326
 - write_bas_head (parameter), 327
 - write_bas_variables (parameter), 327
 - write_data_compressed (parameter), 327
 - write_data_format (parameter), 327
 - write_data_param (parameter), 328
 - write_free_con (parameter), 328
 - write_generic_names (parameter), 328
 - write_generic_names_io (parameter), 329
 - write_int_constraints (parameter), 329
 - write_int_head (parameter), 329
 - write_int_variables (parameter), 329
 - write_lp_gen_var_name (parameter), 341
 - write_lp_line_width (parameter), 330
 - write_lp_quoted_names (parameter), 330
 - write_lp_strict_format (parameter), 330
 - write_lp_terms_per_line (parameter), 330
 - write_mps_int (parameter), 331
 - write_mps_obj_sense (parameter), 331
 - write_mps_quoted_names (parameter), 331
 - write_mps_strict (parameter), 332
 - write_precision (parameter), 332
 - write_sol_constraints (parameter), 332
 - write_sol_head (parameter), 332
 - write_sol_variables (parameter), 333
 - write_task_inc_sol (parameter), 333
 - write_xml_mode (parameter), 333
 - writebranchpriorities (Task method), 215
 - writedata
 - Task.writedata (example), 14
 - writedata (Task method), 215
 - writeparamfile (Task method), 216
 - writesolution (Task method), 216
 - wrn_dropped_nz_qobj (response code), 415
 - wrn_eliminator_space (response code), 415
 - wrn_empty_name (response code), 415
 - wrn_fixed_bound_values (response code), 415
 - wrn_ignore_integer (response code), 416
 - wrn_large_aij (response code), 416
 - wrn_large_bound (response code), 416
 - wrn_large_cj (response code), 416
 - wrn_large_lo_bound (response code), 417
 - wrn_large_up_bound (response code), 417
 - wrn_license_expire (response code), 417
 - wrn_license_feature_expire (response code), 417
 - wrn_license_server (response code), 417
 - wrn_lp_drop_variable (response code), 417
 - wrn_lp_old_quad_format (response code), 418

`wrn_mio_infeasible_final` (response code), 418
`wrn_mps_split_bou_vector` (response code), 418
`wrn_mps_split_ran_vector` (response code), 418
`wrn_mps_split_rhs_vector` (response code), 419
`wrn_name_max_len` (response code), 419
`wrn_no_global_optimizer` (response code), 419
`wrn_noncomplete_linear_dependency_check` (response code), 419
`wrn_nz_in_upr_tri` (response code), 419
`wrn_open_param_file` (response code), 420
`wrn_presolve_bad_precision` (response code), 420
`wrn_presolve_outofspace` (response code), 420
`wrn_sol_filter` (response code), 420
`wrn_spar_max_len` (response code), 420
`wrn_too_few_basis_vars` (response code), 420
`wrn_too_many_basis_vars` (response code), 421
`wrn_undef_sol_file_name` (response code), 421
`wrn_using_generic_names` (response code), 421
`wrn_write_discarded_cfix` (response code), 421
`wrn_zero_aij` (response code), 421
`wrn_zeros_in_sparse_data` (response code), 422

xml format, 487
`mosek.xmlwriteroutputtype`, 454

xx
 `mosek.solitem.xx` (example), 17, 29