MOSEK Optimizer API for Python

Release 9.1.8

MOSEK ApS

15 November 2019

Contents

Chapter 1

Introduction

The **MOSEK** Optimization Suite 9.1.8 is a powerful software package capable of solving large-scale optimization problems of the following kind:

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

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

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

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

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

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

$$Ax - b \geq 0.$$

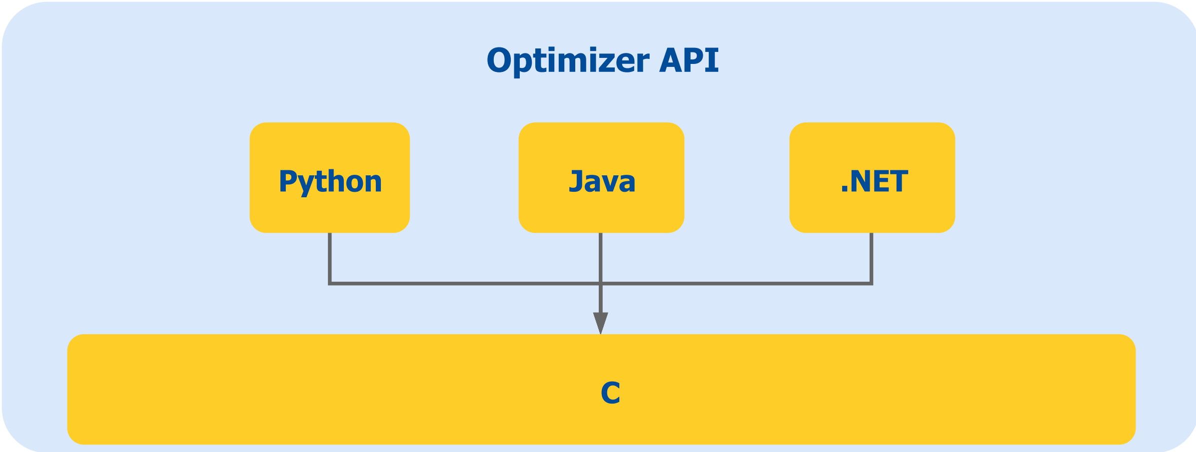
In conic optimization this is replaced with a wider class of constraints

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

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

1.1 Why the Optimizer API for Python?

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



The Optimizer API for Python can be used with any application running on recent Python 2 and 3 interpreters. It consists of a single `mosek` package which can be used in Python scripts and interactive shells making it suited for fast prototyping and inspection of models.

The Optimizer API for Python provides access to:

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

as well as to additional functions for

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

Chapter 2

Contact Information

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

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

Blogger	https://blog.mosek.com/
Google Group	https://groups.google.com/forum/#!forum/mosek
Twitter	https://twitter.com/mosektw
Google+	https://plus.google.com/+Mosek/posts
Linkedin	https://www.linkedin.com/company/mosek-aps

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

Chapter 3

License Agreement

Before using the **MOSEK** software, please read the license agreement available in the distribution at `<MSKHOME>/mosek/9.1/mosek-eula.pdf` or on the **MOSEK** website https://mosek.com/products/license-agreement.

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

zlib

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

Listing 3.1: *zlib* license.

```
zlib.h -- interface of the 'zlib' general purpose compression library
version 1.2.7, May 2nd, 2012

Copyright (C) 1995-2012 Jean-loup Gailly and Mark Adler

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

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

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

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

fplib

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

Listing 3.2: *fplib* license.

```
/****************************************************************
 *
```

(continues on next page)

(continued from previous page)

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

Zstandard

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

Listing 3.3: *Zstandard* license.

```
BSD License

For Zstandard software

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

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

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

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

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

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

Chapter 4

Installation

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

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

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

Compatibility

The Optimizer API for Python requires Python with numpy. The supported versions of Python are shown below:

Platform	Python	PyPy
Linux 64 bit	2.7, 3.6+	2.7
Mac OS 64 bit	2.7, 3.6+	2.7
Windows 32 and 64 bit	2.7, 3.6+	2.7

4.1 Anaconda

The **MOSEK** Optimization Suite can be installed as an Anaconda package, see https://anaconda.org/MOSEK/mosek, for example by running

```
conda install -c mosek mosek
```

If you installed the **MOSEK** package as part of Anaconda, no additional setup is required.

4.2 PIP and Wheels

The **MOSEK** Optimization Suite can be installed as a Wheels package with PIP, using

```
pip install -f https://download.mosek.com/stable/wheel/index.html Mosek --user
```

(skip `--user` for a system-wide installation).

If you installed the **MOSEK** package with PIP, no additional setup is required.

4.3 PyPy

To use **MOSEK** in PyPy install the **MOSEK** Python module from the directory `<PLATFORM>/purepython` instead of `<PLATFORM>/python` as described below.

4.4 Manual installation

Locating files in the MOSEK Optimization Suite

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

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

Relative Path	Description	Label
`<MSKHOME>/mosek/9.1/tools/platform/<PLATFORM>/python/2`	Python 2 install	`<PYTHON2DIR>`
`<MSKHOME>/mosek/9.1/tools/platform/<PLATFORM>/python/3`	Python 3 install	`<PYTHON3DIR>`
`<MSKHOME>/mosek/9.1/tools/examples/python`	Examples	`<EXDIR>`
`<MSKHOME>/mosek/9.1/tools/examples/data`	Additional data	`<MISCDIR>`

where

- `<MSKHOME>` is the folder in which the **MOSEK** Optimization Suite has been installed,
- `<PLATFORM>` is the actual platform among those supported by **MOSEK**, i.e. `win32x86`, `win64x86`, `linux64x86` or `osx64x86`.

Manual install and setting up paths

To install **MOSEK** for Python run the `<PYTHON2DIR>/setup.py` or `<PYTHON3DIR>/setup.py` script depending on the Python version you want to use. This will add the **MOSEK** module to your Python distribution's library of modules. The script accepts the standard options typical for Python setup scripts. For instance, to install **MOSEK** for Python 3 in the user's local library run:

```
$ python3 <PYTHON3DIR>/setup.py install --user
```

on Linux and Mac OS or

```
C:\> python3 <PYTHON3DIR>\setup.py install --user
```

on Windows.
For a system-wide installation drop the `--user` flag.

4.5 Testing the Installation

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

```python
import mosek
```

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

```
python lo1.py
```

Chapter 5

Design Overview

5.1 Modeling

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

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

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

The main characteristics of this interface are:

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

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

5.2 "Hello World!" in MOSEK

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

Creating an environment and task

Every interaction with **MOSEK** using Optimizer API for Python begins by creating a **MOSEK environment**. It coordinates the access to **MOSEK** from the current process.

In most cases the user does not interact directly with the environment, except for creating optimization **tasks**, which contain actual problem specifications and where optimization takes place. An environment can host multiple tasks.

Defining tasks

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

Retrieving the solutions

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

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

Source code example

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

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

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

Listing 5.1: "Hello World!" in MOSEK

```
from mosek import *;

x = [ 0.0 ]

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

    task.optimize()                         # Optimize

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

Chapter 6

Optimization Tutorials

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

6.1 Linear Optimization

The simplest optimization problem is a purely linear problem. A *linear optimization problem* is a problem of the following form:

Minimize or maximize the objective function

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

subject to the linear constraints

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

and the bounds

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

The problem description consists of the following elements:

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

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

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

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

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

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

6.1.1 Example LO1

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

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

under the bounds

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

Solving the problem

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

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

Below we explain each of these steps.

Create an environment.

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

```
# Make mosek environment
with mosek.Env() as env:
```

Create an optimization task.

Next, an empty task object is created:

```
# Create a task object
with env.Task(0, 0) as task:
    # Attach a log stream printer to the task
    task.set_Stream(mosek.streamtype.log, streamprinter)
```

We also connect a call-back function to the task log stream. Messages related to the task are passed to the call-back function. In this case the stream call-back function writes its messages to the standard output stream. See Sec. 7.3.

Load a problem into the task object.

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

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

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

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

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

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

Setting bounds on variables

The bounds on variables are stored in the arrays

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

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

and are set with calls to *Task.putvarbound*.

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

The *Bound key* stored in bkx specifies the type of the bound according to Table 6.1.

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

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

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

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

and

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

Defining the linear constraint matrix.

Recall that in our example the A matrix is given by

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

This matrix is stored in sparse format in the arrays:

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

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

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

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

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

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

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

Optimization

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

```
task.optimize()
```

Extracting the solution.

After optimizing the status of the solution is examined with a call to *Task.getsolsta*. If the solution status is reported as *solsta.optimal* the solution is extracted in the lines below:

```
    xx = [0.] * numvar
    task.getxx(mosek.soltype.bas, # Request the basic solution.
               xx)
```

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

Catching exceptions

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

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

The types of exceptions that **MOSEK** can throw can be seen in Sec. 15.5. See also Sec. 7.2.

Source code

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

Listing 6.1: Linear optimization example.

```
import sys
import mosek

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

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


def main():
    # Make mosek environment
    with mosek.Env() as env:
        # Create a task object
        with env.Task(0, 0) as task:
            # Attach a log stream printer to the task
            task.set_Stream(mosek.streamtype.log, streamprinter)

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

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

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

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

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

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

            numvar = len(bkx)
```

(continues on next page)

(continued from previous page)

```
            numcon = len(bkc)

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

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

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

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

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

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

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

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

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

            if (solsta == mosek.solsta.optimal):
                xx = [0.] * numvar
                task.getxx(mosek.soltype.bas, # Request the basic solution.
                           xx)
                print("Optimal solution: ")
                for i in range(numvar):
                    print("x[" + str(i) + "]=" + str(xx[i]))
            elif (solsta == mosek.solsta.dual_infeas_cer or
                  solsta == mosek.solsta.prim_infeas_cer):
                print("Primal or dual infeasibility certificate found.\n")
            elif solsta == mosek.solsta.unknown:
                print("Unknown solution status")
            else:
                print("Other solution status")

# call the main function
try:
    main()
except mosek.Error as e:
    print("ERROR: %s" % str(e.errno))
    if e.msg is not None:
```

(continues on next page)

(continued from previous page)

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

6.2 Quadratic Optimization

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

$$\begin{array}{lccccl}\text{minimize} & & \frac{1}{2}x^TQ^ox + c^Tx + c^f & & & \\ \text{subject to} & l_k^c \leq & \frac{1}{2}x^TQ^kx + \sum_{j=0}^{n-1} a_{k,j}x_j & \leq & u_k^c, & k = 0,\ldots,m-1, \\ & l_j^x \leq & x_j & \leq & u_j^x, & j = 0,\ldots,n-1.\end{array} \tag{6.2}$$

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

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

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

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

$$l_k^c \leq \frac{1}{2}x^TQ^kx + \sum_{j=0}^{n-1} a_{k,j}x_j \tag{6.3}$$

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

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

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

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

$$x^TQx \geq 0, \quad \forall x.$$

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

6.2.1 Example: Quadratic Objective

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

$$\begin{array}{lll}\text{minimize} & & x_1^2 + 0.1x_2^2 + x_3^2 - x_1x_3 - x_2 \\ \text{subject to} & 1 \leq & x_1 + x_2 + x_3 \\ & 0 \leq & x.\end{array} \tag{6.4}$$

The matrix formulation of (6.4) has:

$$Q^o = \left[\begin{array}{ccc} 2 & 0 & -1 \\ 0 & 0.2 & 0 \\ -1 & 0 & 2 \end{array}\right], c = \left[\begin{array}{c} 0 \\ -1 \\ 0 \end{array}\right], A = \left[\begin{array}{ccc} 1 & 1 & 1 \end{array}\right],$$

with the bounds:

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

Please note the explicit $\frac{1}{2}$ in the objective function of (6.2) which implies that diagonal elements must be doubled in Q, i.e. $Q_{11} = 2$ even though 1 is the coefficient in front of x_1^2 in (6.4).

Setting up the linear part

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

Setting up the quadratic objective

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

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

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

Please note that

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

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

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

Source code

Listing 6.2: Source code implementing problem (6.4).

```
import sys, os, mosek

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

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


def main():
    # Open MOSEK and create an environment and task
    # Make a MOSEK environment
    with mosek.Env() as env:
        # Attach a printer to the environment
        env.set_Stream(mosek.streamtype.log, streamprinter)
```

(continues on next page)

(continued from previous page)

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

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

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

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

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

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

            task.putqobj(qsubi, qsubj, qval)

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

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

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

            # Output a solution
            xx = [0.] * numvar
```

(continues on next page)

(continued from previous page)

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

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

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

6.2.2 Example: Quadratic constraints

In this section we show how to solve a problem with quadratic constraints. Please note that quadratic constraints are subject to the convexity requirement (6.3).

Consider the problem:

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

This is equivalent to

$$\begin{array}{lccc} \text{minimize} & \frac{1}{2}x^TQ^ox + c^Tx & & \\ \text{subject to} & \frac{1}{2}x^TQ^0x + Ax & \geq & b, \\ & x \geq 0, & & \end{array} \tag{6.5}$$

where

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

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

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

Setting up quadratic constraints

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

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

            # put Q^0 in constraint with index 0.

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

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

Source code

Listing 6.3: Implementation of the quadratically constrained problem (6.5).

```
import sys
import mosek

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

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


def main():
    # Make a MOSEK environment
    with mosek.Env() as env:
        # Attach a printer to the environment
        env.set_Stream(mosek.streamtype.log, streamprinter)

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

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

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

            c = [0.0, -1.0, 0.0]

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

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

(continues on next page)

(continued from previous page)

```
        task.appendcons(numcon)

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

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

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

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

        # Set up and input quadratic objective

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

        task.putqobj(qsubi, qsubj, qval)

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

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

        # put Q^0 in constraint with index 0.

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

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

        # Optimize the task
        task.optimize()

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

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

        # Output a solution
        xx = [0.] * numvar
        task.getxx(mosek.soltype.itr,
```

(continues on next page)

(continued from previous page)

```
                       xx)

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

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

6.3 Conic Quadratic Optimization

Conic optimization is a generalization of linear optimization, allowing constraints of the type

$$x^t \in \mathcal{K}_t,$$

where x^t is a subset of the problem variables and $\mathcal{K}_t$ is a convex cone. Since the set $\mathbb{R}^n$ of real numbers is also a convex cone, we can simply write a compound conic constraint $x \in \mathcal{K}$ where $\mathcal{K} = \mathcal{K}_1 \times \cdots \times \mathcal{K}_l$ is a product of smaller cones and x is the full problem variable.

MOSEK can solve conic quadratic optimization problems of the form

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

where the domain restriction, $x \in \mathcal{K}$, implies that all variables are partitioned into convex cones

$$x = (x^0, x^1, \ldots, x^{p-1}), \quad \text{with } x^t \in \mathcal{K}_t \subseteq \mathbb{R}^{n_t}.$$

In this tutorial we describe how to use the two types of quadratic cones defined as:

- Quadratic cone:

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

- Rotated quadratic cone:

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

For other types of cones supported by **MOSEK** see Sec. 6.4, Sec. 6.5, Sec. 6.6. See *Task.appendcone* for a list and definitions of available cone types. Different cone types can appear together in one optimization problem.

For example, the following constraint:

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

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

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

Furthermore, each variable may belong to one cone at most. The constraint $x_i - x_j = 0$ would however allow x_i and x_j to belong to different cones with same effect.

6.3.1 Example CQO1

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

$$\begin{array}{lccl}
\text{minimize} & x_4 + x_5 + x_6 & & \\
\text{subject to} & x_1 + x_2 + 2x_3 & = & 1, \\
 & x_1, x_2, x_3 & \geq & 0, \\
 & x_4 \geq \sqrt{x_1^2 + x_2^2}, & & \\
 & 2x_5 x_6 \geq x_3^2 & &
\end{array} \tag{6.6}$$

Setting up the linear part

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

Setting up the conic constraints

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

```
            task.appendcone(mosek.conetype.quad,
                            0.0,
                            [3, 0, 1])
```

The first argument selects the type of quadratic cone, in this case either *conetype.quad* for a *quadratic cone* or *conetype.rquad* for a *rotated quadratic cone*. The second parameter is currently ignored and passing `0.0` will work.

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

Variants of this method are available to append multiple cones at a time.

Source code

Listing 6.4: Source code solving problem (6.6).

```
import sys
import mosek

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

# Define a stream printer to grab output from MOSEK
def streamprinter(text):
```

(continues on next page)

(continued from previous page)

```python
    sys.stdout.write(text)
    sys.stdout.flush()


def main():
    # Make a MOSEK environment
    with mosek.Env() as env:
        # Attach a printer to the environment
        env.set_Stream(mosek.streamtype.log, streamprinter)

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

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

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

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

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

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

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

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

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

                # Input the cones
            task.appendcone(mosek.conetype.quad,
```

(continues on next page)

(continued from previous page)

```
                        0.0,
                        [3, 0, 1])
            task.appendcone(mosek.conetype.rquad,
                        0.0,
                        [4, 5, 2])

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

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

            # Output a solution
            xx = [0.] * numvar
            task.getxx(mosek.soltype.itr,
                       xx)

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

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

6.4 Power Cone Optimization

Conic optimization is a generalization of linear optimization, allowing constraints of the type

$$x^t \in \mathcal{K}_t,$$

where x^t is a subset of the problem variables and $\mathcal{K}_t$ is a convex cone. Since the set $\mathbb{R}^n$ of real numbers is also a convex cone, we can simply write a compound conic constraint $x \in \mathcal{K}$ where $\mathcal{K} = \mathcal{K}_1 \times \cdots \times \mathcal{K}_l$ is a product of smaller cones and x is the full problem variable.

MOSEK can solve conic optimization problems of the form

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

where the domain restriction, $x \in \mathcal{K}$, implies that all variables are partitioned into convex cones

$$x = (x^0, x^1, \ldots, x^{p-1}), \quad \text{with } x^t \in \mathcal{K}_t \subseteq \mathbb{R}^{n_t}.$$

In this tutorial we describe how to use the power cone. The primal power cone of dimension n with parameter $0 < \alpha < 1$ is defined as:

$$\mathcal{P}_n^{\alpha,1-\alpha} = \left\{ x \in \mathbb{R}^n : x_0^\alpha x_1^{1-\alpha} \geq \sqrt{\sum_{i=2}^{n-1} x_i^2},\ x_0, x_1 \geq 0 \right\}.$$

In particular, the most important special case is the three-dimensional power cone family:

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

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

MOSEK also supports the dual power cone:

$$\left(\mathcal{P}_n^{\alpha,1-\alpha}\right)^* = \left\{ x \in \mathbb{R}^n : \left(\frac{x_0}{\alpha}\right)^\alpha \left(\frac{x_1}{1-\alpha}\right)^{1-\alpha} \geq \sqrt{\sum_{i=2}^{n-1} x_i^2},\ x_0, x_1 \geq 0 \right\}.$$

For other types of cones supported by **MOSEK** see Sec. 6.3, Sec. 6.5, Sec. 6.6. See *Task.appendcone* for a list and definitions of available cone types. Different cone types can appear together in one optimization problem.

Furthermore, each variable may belong to one cone at most. The constraint $x_i - x_j = 0$ would however allow x_i and x_j to belong to different cones with same effect.

6.4.1 Example POW1

Consider the following optimization problem which involves powers of variables:

$$\begin{array}{lrcl} \text{maximize} & x^{0.2} y^{0.8} + z^{0.4} - x & & \\ \text{subject to} & x + y + \frac{1}{2} z & = & 2, \\ & x, y, z & \geq & 0. \end{array} \tag{6.7}$$

With $(x, y, z) = (x_0, x_1, x_2)$ we convert it into conic form using auxiliary variables as bounds for the power expressions:

$$\begin{array}{lrcl} \text{maximize} & x_3 + x_4 - x_0 & & \\ \text{subject to} & x_0 + x_1 + \frac{1}{2} x_2 & = & 2, \\ & (x_0, x_1, x_3) & \in & \mathcal{P}_3^{0.2,0.8}, \\ & (x_2, x_5, x_4) & \in & \mathcal{P}_3^{0.4,0.6}, \\ & x_5 & = & 1. \end{array} \tag{6.8}$$

Setting up the linear part

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

Setting up the conic constraints

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

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

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

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

```
[ 0.06389298  0.78308564  2.30604283 ]
```

Source code

Listing 6.5: Source code solving problem (6.7).

```
import sys
import mosek

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

def main():

    # Only a symbolic constant
    inf = 0.0

    # Make a MOSEK environment
    with mosek.Env() as env:
        # Attach a printer to the environment
        env.set_Stream(mosek.streamtype.log, streamprinter)

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

            csub = [3, 4, 0]
            cval = [1.0, 1.0, -1.0]
            asub = [0, 1, 2]
            aval = [1.0, 1.0, 0.5]
            numvar, numcon = 6, 1

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

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

            # Set up the linear part of the problem
            task.putclist(csub, cval)
            task.putarow(0, asub, aval)
            task.putvarboundslice(0, numvar, [mosek.boundkey.fr] * numvar, [inf] * numvar,␣
↪[inf] * numvar)
            task.putvarbound(5, mosek.boundkey.fx, 1.0, 1.0) # x_5 = 1
            task.putconbound(0, mosek.boundkey.fx, 2.0, 2.0)

            # Input the cones
            task.appendcone(mosek.conetype.ppow, 0.2, [0, 1, 3])
            task.appendcone(mosek.conetype.ppow, 0.4, [2, 5, 4])

            # Input the objective sense (minimize/maximize)
```

(continues on next page)

(continued from previous page)

```
            task.putobjsense(mosek.objsense.maximize)

            # Optimize the task
            task.optimize()

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

            # Output a solution
            xx = [0.] * numvar
            task.getxx(mosek.soltype.itr,
                       xx)

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

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

6.5 Conic Exponential Optimization

Conic optimization is a generalization of linear optimization, allowing constraints of the type

$$x^t \in \mathcal{K}_t,$$

where x^t is a subset of the problem variables and $\mathcal{K}_t$ is a convex cone. Since the set $\mathbb{R}^n$ of real numbers is also a convex cone, we can simply write a compound conic constraint $x \in \mathcal{K}$ where $\mathcal{K} = \mathcal{K}_1 \times \cdots \times \mathcal{K}_l$ is a product of smaller cones and x is the full problem variable.

MOSEK can solve conic optimization problems of the form

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

where the domain restriction, $x \in \mathcal{K}$, implies that all variables are partitioned into convex cones

$$x = (x^0, x^1, \ldots, x^{p-1}), \quad \text{with } x^t \in \mathcal{K}_t \subseteq \mathbb{R}^{n_t}.$$

In this tutorial we describe how to use the primal exponential cone defined as:

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

MOSEK also supports the dual exponential cone:

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

For other types of cones supported by **MOSEK** see Sec. 6.3, Sec. 6.4, Sec. 6.6. See *Task.appendcone* for a list and definitions of available cone types. Different cone types can appear together in one optimization problem.

For example, the following constraint:

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

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

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

Furthermore, each variable may belong to one cone at most. The constraint $x_i - x_j = 0$ would however allow x_i and x_j to belong to different cones with same effect.

6.5.1 Example CEO1

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

$$\begin{array}{lrcl}
\text{minimize} & x_0 + x_1 & & \\
\text{subject to} & x_0 + x_1 + x_2 & = & 1, \\
 & x_0 & \geq & x_1 \exp(x_2/x_1), \\
 & x_0, x_1 & \geq & 0.
\end{array} \tag{6.9}$$

The conic form of (6.9) is:

$$\begin{array}{lrcl}
\text{minimize} & x_0 + x_1 & & \\
\text{subject to} & x_0 + x_1 + x_2 & = & 1, \\
 & (x_0, x_1, x_2) & \in & K_{\exp}, \\
 & x & \in & \mathbb{R}^3.
\end{array} \tag{6.10}$$

Setting up the linear part

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

Setting up the conic constraints

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

```
        task.appendcone(mosek.conetype.pexp,
                        0.0,
                        [0, 1, 2])
```

The first argument selects the type of exponential cone, that is *conetype.pexp*. The second parameter is currently ignored and passing `0.0` will work.

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

Variants of this method are available to append multiple cones at a time.

Source code

Listing 6.6: Source code solving problem (6.9).

```
import sys
import mosek

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

def main():

    # Only a symbolic constant
    inf = 0.0

    # Make a MOSEK environment
    with mosek.Env() as env:
        # Attach a printer to the environment
        env.set_Stream(mosek.streamtype.log, streamprinter)

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

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

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

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

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

            # Input the cones
            task.appendcone(mosek.conetype.pexp,
                            0.0,
                            [0, 1, 2])

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

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

(continues on next page)

```
            # Output a solution
            xx = [0.] * numvar
            task.getxx(mosek.soltype.itr,
                       xx)

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

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

6.6 Semidefinite Optimization

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

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

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

MOSEK can solve semidefinite optimization problems of the form

$$\begin{array}{lrcccl} \text{minimize} & & & \sum_{j=0}^{n-1} c_j x_j + \sum_{j=0}^{p-1} \left\langle \overline{C}_j, \overline{X}_j \right\rangle + c^f & & \\ \text{subject to} & l_i^c & \leq & \sum_{j=0}^{n-1} a_{ij} x_j + \sum_{j=0}^{p-1} \left\langle \overline{A}_{ij}, \overline{X}_j \right\rangle & \leq & u_i^c, \quad i = 0, \ldots, m-1, \\ & l_j^x & \leq & x_j & \leq & u_j^x, \quad j = 0, \ldots, n-1, \\ & & & x \in \mathcal{K}, \overline{X}_j \in \mathcal{S}_+^{r_j}, & & \quad j = 0, \ldots, p-1 \end{array}$$

where the problem has p symmetric positive semidefinite variables $\overline{X}_j \in \mathcal{S}_+^{r_j}$ of dimension r_j with symmetric coefficient matrices $\overline{C}_j \in \mathcal{S}^{r_j}$ and $\overline{A}_{i,j} \in \mathcal{S}^{r_j}$. We use standard notation for the matrix inner product, i.e., for $A, B \in \mathbb{R}^{m \times n}$ we have

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

6.6.1 Example SDO1

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

$$
\begin{array}{llcl}
\text{minimize} & \left\langle \begin{bmatrix} 2 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 2 \end{bmatrix}, \overline{X} \right\rangle + x_0 & & \\
\text{subject to} & \left\langle \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \overline{X} \right\rangle + x_0 & = & 1, \\
& \left\langle \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \overline{X} \right\rangle + x_1 + x_2 & = & 1/2, \\
& x_0 \geq \sqrt{{x_1}^2 + {x_2}^2}, & & \overline{X} \succeq 0,
\end{array} \tag{6.11}
$$

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

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

and a conic quadratic variable $(x_0, x_1, x_2) \in \mathcal{Q}^3$. The objective is to minimize

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

subject to the two linear constraints

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

Setting up the linear and conic part

The linear and conic parts (constraints, variables, objective, cones) are set up using the methods described in the relevant tutorials; Sec. 6.1, Sec. 6.3, Sec. 6.5, Sec. 6.4. Here we only discuss the aspects directly involving semidefinite variables.

Appending semidefinite variables

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

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

Appending coefficient matrices

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

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

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

            syma1 = task.appendsparsesymmat(BARVARDIM[0],
```

(continues on next page)

(continued from previous page)

```
                              barai[1],
                              baraj[1],
                              baraval[1])
```

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

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

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

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

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

Retrieving the solution

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

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

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

Source code

Listing 6.7: Source code solving problem (6.11).

```
import sys
import mosek

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

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


def main():
    # Make mosek environment
    with mosek.Env() as env:

        # Create a task object and attach log stream printer
        with env.Task(0, 0) as task:
            task.set_Stream(mosek.streamtype.log, streamprinter)

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

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

(continues on next page)

(continued from previous page)

```
buc = [1.0, 0.5]

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

conesub = [0, 1, 2]

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

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

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

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

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

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

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

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

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

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

# Add the quadratic cone constraint
task.appendcone(mosek.conetype.quad,
                0.0,
                conesub)
```

(continues on next page)

(continued from previous page)

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

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

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

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

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

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

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

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

            if (solsta == mosek.solsta.optimal):
                xx = [0.] * numvar
                task.getxx(mosek.soltype.itr, xx)

                lenbarvar = BARVARDIM[0] * (BARVARDIM[0] + 1) / 2
                barx = [0.] * int(lenbarvar)
                task.getbarxj(mosek.soltype.itr, 0, barx)

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

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

6.7 Integer Optimization

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

6.7.1 Example MILO1

We use the example

$$
\begin{array}{lccl}
\text{maximize} & x_0 + 0.64 x_1 & & \\
\text{subject to} & 50 x_0 + 31 x_1 & \leq & 250, \\
 & 3 x_0 - 2 x_1 & \geq & -4, \\
 & x_0, x_1 \geq 0 & & \text{and integer}
\end{array}
\tag{6.12}
$$

to demonstrate how to set up and solve a problem with integer variables. It has the structure of a linear optimization problem (see Sec. 6.1) except for integrality constraints on the variables. Therefore, only the specification of the integer constraints requires something new compared to the linear optimization problem discussed previously.

First, the integrality constraints are imposed using the function *Task.putvartype*:

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

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

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

The complete source for the example is listed Listing 6.8. Please note that when *Task.getsolutionslice* is called, the integer solution is requested by using *soltype.itg*. No dual solution is defined for integer optimization problems.

Listing 6.8: Source code implementing problem (6.12).

```
import sys
import mosek

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

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


def main():
    # Make a MOSEK environment
    with mosek.Env() as env:
        # Attach a printer to the environment
        env.set_Stream(mosek.streamtype.log, streamprinter)

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

```

(continues on next page)

(continued from previous page)

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

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

        c = [1.0, 0.64]

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

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

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

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

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

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

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

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

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


        # Optimize the task
        task.optimize()

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

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

        # Output a solution
```

(continues on next page)

(continued from previous page)

```
            xx = [0.] * numvar
            task.getxx(mosek.soltype.itg, xx)

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

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

6.7.2 Specifying an initial solution

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

It is not necessary to specify the whole solution. **MOSEK** will attempt to use it to speed up the computation. **MOSEK** will first try to construct a feasible solution by fixing integer variables to the values provided by the user (rounding if necessary) and optimizing over the continuous variables. The outcome of this process can be inspected via information items *iinfitem.mio_construct_solution* and *dinfitem.mio_construct_solution_obj*, and via the `Construct solution objective` entry in the log. We concentrate on a simple example below.

$$
\begin{array}{ll}
\text{maximize} & 7x_0 + 10x_1 + x_2 + 5x_3 \\
\text{subject to} & x_0 + x_1 + x_2 + x_3 \leq 2.5 \\
 & x_0, x_1, x_2 \in \mathbb{Z} \\
 & x_0, x_1, x_2, x_3 \geq 0
\end{array} \tag{6.13}
$$

Solution values can be set using *Task.putsolution* .

Listing 6.9: Implementation of problem (6.13) specifying an initial solution.

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

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

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

The same information can be obtained from the API:

Listing 6.10: Retrieving information about usage of initial solution

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

6.7.3 Example MICO1

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

$$\begin{array}{ll} \text{minimize} & x^2+y^2 \\ \text{subject to} & x \geq e^y + 3.8, \\ & x, y \text{ integer}. \end{array} \tag{6.14}$$

The canonical conic formulation of (6.14) suitable for Optimizer API for Python is

$$\begin{array}{lrcll} \text{minimize} & x_0 & & & \\ \text{subject to} & (x_0, x_1, x_2) & \in & \mathcal{Q}^3 & (x_0 \geq \sqrt{x_1^2 + x_2^2}) \\ & (x_3, x_4, x_5) & \in & K_{\exp} & (x_3 \geq x_4 \exp(x_5/x_4)) \\ & -x_1 + x_3 & = & -3.8 & \\ & x_4 & = & 1 & \\ & x_2 - x_5 & = & 0 & \\ & x_1, x_2 & & \text{integer}. & \end{array} \tag{6.15}$$

Listing 6.11: Implementation of problem (6.15).

```
with mosek.Env() as env:
    with env.Task(0, 0) as task:
        task.set_Stream(mosek.streamtype.log, streamprinter)

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

        # Integrality constraints
        task.putvartypelist([1,2], [mosek.variabletype.type_int]*2)

        # Set up the three auxiliary linear constraints
        task.putaijlist([0,0,1,2,2],
                        [1,3,4,2,5],
                        [-1,1,1,1,-1])
        task.putconboundslice(0, 3, [mosek.boundkey.fx]*3, [-3.8, 1, 0], [-3.8, 1, 0])

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

        # Conic part of the problem
        task.appendconesseq([mosek.conetype.quad, mosek.conetype.pexp], [0, 0], [3, 3], 0)

        # Optimize the task
        task.optimize()
        task.solutionsummary(mosek.streamtype.msg)
```

(continues on next page)

(continued from previous page)

```
    xx = [0, 0]
    task.getxxslice(mosek.soltype.itg, 1, 3, xx)
    print(xx)
```

Error and solution status handling were omitted for readability.

6.8 Geometric Programming

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

$$\begin{array}{lll} \text{minimize} & f_0(x) & \\ \text{subject to} & f_i(x) \leq 1, & i = 1, \ldots, m, \\ & x_j > 0, & j = 1, \ldots, n, \end{array} \tag{6.16}$$

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

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

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

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

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

$$\log(\sum_k \exp(a_k^T y + b_k)) \leq 0 \tag{6.17}$$

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

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

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

6.8.1 Example GP1

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

$$\begin{array}{lrl} \text{maximize} & hwd & \\ \text{subject to} & 2(hw + hd) & \leq A_{\text{wall}}, \\ & wd & \leq A_{\text{floor}}, \\ & \alpha & \leq h/w \leq \beta, \\ & \gamma & \leq d/w \leq \delta. \end{array} \tag{6.18}$$

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

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

after which (6.18) becomes

$$\begin{array}{ll} \text{maximize} & x + y + z \\ \text{subject to} & \log(\exp(x + y + \log(2/A_{\text{wall}})) + \exp(x + z + \log(2/A_{\text{wall}}))) \leq 0, \\ & y + z \leq \log(A_{\text{floor}}), \\ & \log(\alpha) \leq x - y \leq \log(\beta), \\ & \log(\gamma) \leq z - y \leq \log(\delta). \end{array} \tag{6.19}$$

Next, we demonstrate how to implement a log-sum-exp constraint (6.17). It can be written as:

$$\begin{array}{ll} u_k \geq \exp(a_k^T y + b_k), & (\text{equiv. } (u_k, 1, a_k^T y + b_k) \in K_{\exp}), \\ \sum_k u_k = 1. & \end{array} \tag{6.20}$$

This presentation requires one extra variable u_k for each monomial appearing in the original posynomial constraint. Another fixed variable $t_k = 1$ stands for the second entry in the exponential cone.

Listing 6.12: Implementation of log-sum-exp as in (6.20).

```
        # Add a single log-sum-exp constraint sum(log(exp(z_i))) <= 0
        # Assume numExp variable triples are ordered as (u0,t0,z0,u1,t1,z1...)
        # starting from variable with index expStart

        # sum(u_i) = 1 as constraint number c, u_i unbounded
        task.putarow(c, range(expStart, expStart + 3*numExp, 3), [1.0]*numExp)
        task.putconbound(c, boundkey.fx, 1.0, 1.0)
        task.putvarboundlistconst(range(expStart, expStart + 3*numExp, 3),
                                  boundkey.fr, -inf, inf)

        # z_i unbounded
        task.putvarboundlistconst(range(expStart + 2, expStart + 2 + 3*numExp, 3),
                                  boundkey.fr, -inf, inf)

        # t_i = 1
        task.putvarboundlistconst(range(expStart + 1, expStart + 1 + 3*numExp, 3),
                                  boundkey.fx, 1.0, 1.0)

        # Every triple is in an exponential cone
        task.appendconesseq([conetype.pexp]*numExp, [0.0]*numExp, [3]*numExp, expStart)
```

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

Listing 6.13: Source code solving problem (6.19).

```
def max_volume_box(Aw, Af, alpha, beta, gamma, delta):
    # Basic dimensions of our problem
    numvar    = 3  # Variables in original problem
    numLinCon = 3  # Linear constraints in original problem
    numExp    = 2  # Number of exp-terms in the log-sum-exp constraint

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

    # Linear part setting up slack variables
    # for the linear expressions appearing inside exps
    # x_5 - x - y = log(2/Awall)
    # x_8 - x - z = log(2/Awall)
    # The slack indexes are convenient for defining exponential cones, see later
    a2subi = [3, 3, 3, 4, 4, 4]
    a2subj = [5, 0, 1, 8, 0, 2]
    a2val  = [1.0, -1.0, -1.0, 1.0, -1.0, -1.0]
    b2kc   = [boundkey.fx, boundkey.fx]
    b2luc  = [log(2/Aw), log(2/Aw)]

    with Env() as env:
```

(continues on next page)

(continued from previous page)

```
        with env.Task(0, 0) as task:
            task.set_Stream(streamtype.log, streamprinter)

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

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

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

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

            c = numLinCon + numExp
            expStart = numvar
            # Add a single log-sum-exp constraint sum(log(exp(z_i))) <= 0
            # Assume numExp variable triples are ordered as (u0,t0,z0,u1,t1,z1...)
            # starting from variable with index expStart

            # sum(u_i) = 1 as constraint number c, u_i unbounded
            task.putarow(c, range(expStart, expStart + 3*numExp, 3), [1.0]*numExp)
            task.putconbound(c, boundkey.fx, 1.0, 1.0)
            task.putvarboundlistconst(range(expStart, expStart + 3*numExp, 3),
                                      boundkey.fr, -inf, inf)

            # z_i unbounded
            task.putvarboundlistconst(range(expStart + 2, expStart + 2 + 3*numExp, 3),
                                      boundkey.fr, -inf, inf)

            # t_i = 1
            task.putvarboundlistconst(range(expStart + 1, expStart + 1 + 3*numExp, 3),
                                      boundkey.fx, 1.0, 1.0)

            # Every triple is in an exponential cone
            task.appendconesseq([conetype.pexp]*numExp, [0.0]*numExp, [3]*numExp, expStart)

            # Solve and map to original h, w, d
            task.optimize()
            xyz = [0.0]*numvar
            task.getxxslice(soltype.itr, 0, numvar, xyz)
            return exp(xyz)
```

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

Listing 6.14: Sample data for problem (6.18).

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

6.9 Library of basic functions

This section contains a library of small models of basic functions frequently appearing in optimization models. It is essentially an implementation of the mathematical models from the **MOSEK** Modeling

Cookbook using Optimizer API for Python. These short code snippets can be seen as illustrative examples, can be copy-pasted to other code, and can even be directly called when assembling optimization models as we show in Sec. 6.9.6 (although this may be more suitable for prototyping; also note that additional variables and constraints will be introduced and there is no error checking).

6.9.1 Variable and constraint management

Append variables

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

Listing 6.15: New variables.

```
def msk_newvar(task, num):  # free
    v = task.getnumvar()
    task.appendvars(num)
    for i in range(num):
        task.putvarbound(v+i, mosek.boundkey.fr, -inf, inf)
    return v
def msk_newvar_fx(task, num, val):  # fixed
    v = task.getnumvar()
    task.appendvars(num)
    for i in range(num):
        task.putvarbound(v+i, mosek.boundkey.fx, val, val)
    return v
def msk_newvar_bin(task, num):  # binary
    v = task.getnumvar()
    task.appendvars(num)
    for i in range(num):
        task.putvarbound(v+i, mosek.boundkey.ra, 0.0, 1.0)
        task.putvartype(v+i, mosek.variabletype.type_int)
    return v
```

Variable duplication

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

Listing 6.16: Duplicate variables.

```
# x = y
def msk_equal(task, x, y):
    c = msk_newcon(task, 1)
    task.putaij(c, x, 1.0)
    task.putaij(c, y, -1.0)
    task.putconbound(c, mosek.boundkey.fx, 0.0, 0.0)
def msk_dup(task, x):
    y = msk_newvar(task, 1)
    msk_equal(task, x, y)
    return y
```

Append constraints

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

Listing 6.17: New constraints.

```
def msk_newcon(task, num):
    c = task.getnumcon()
    task.appendcons(num)
    return c
```

6.9.2 Linear operations

Absolute value

$t \geq |x|$

Listing 6.18: Absolute value.

```
# t >= |x|
def msk_abs(task, t, x):
    c = msk_newcon(task, 2)
    task.putaij(c, t, 1.0)
    task.putaij(c, x, 1.0)
    task.putconbound(c, mosek.boundkey.lo, 0.0, inf)
    task.putaij(c+1, t, 1.0)
    task.putaij(c+1, x, -1.0)
    task.putconbound(c+1, mosek.boundkey.lo, 0.0, inf)
```

1-norm

$t \geq \sum_i |x_i|$

Listing 6.19: 1-norm.

```
# t >= sum( |x_i| ), x is a list of variables
def msk_norm1(task, t, x):
    n = len(x)
    u = msk_newvar(task, n)
    for i in range(n):
        msk_abs(task, u+i, x[i])
    c = msk_newcon(task, 1)
    task.putarow(c, range(u, u+n), [-1.0]*n)
    task.putaij(c, t, 1.0)
    task.putconbound(c, mosek.boundkey.lo, 0.0, inf)
```

6.9.3 Quadratic and power operations

Square

$t \geq x^2$

Listing 6.20: Square.

```
# t >= x^2
def msk_sq(task, t, x):
    task.appendcone(mosek.conetype.rquad, 0.0, [msk_newvar_fx(task, 1, 0.5), t, x])
```

2-norm

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

Listing 6.21: 2-norm.

```
# t >= sqrt(x_1^2 + ... + x_n^2) where x is a list of variables
def msk_norm2(task, t, x):
    task.appendcone(mosek.conetype.quad, 0.0, [t] + x)
```

Powers

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

Listing 6.22: Power.

```
# t >= |x|^p (where p>1)
def msk_pow(task, t, x, p):
    task.appendcone(mosek.conetype.ppow, 1.0/p, [t, msk_newvar_fx(task, 1, 1.0), x])
```

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

Listing 6.23: Power reciprocal.

```
# t >= 1/x^p, x>0 (where p>0)
def msk_pow_inv(task, t, x, p):
    task.appendcone(mosek.conetype.ppow, 1.0/(1.0+p), [t, x, msk_newvar_fx(task, 1, 1.0)])
```

p-norm

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

Listing 6.24: p-norm.

```
# t >= \|x\|_p (where p>1), x is a list of variables
def msk_pnorm(task, t, x, p):
    n = len(x)
    r = msk_newvar(task, n)
    for i in range(n):
        task.appendcone(mosek.conetype.ppow, 1.0-1.0/p, [t, r+i, x[i]])
    c = msk_newcon(task, 1)
    task.putarow(c, range(r, r+n), [-1.0]*n)
    task.putaij(c, t, 1.0)
    task.putconbound(c, mosek.boundkey.fx, 0.0, 0.0)
```

Geometric mean

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

Listing 6.25: Geometric mean.

```
# |t| <= (x_1...x_n)^(1/n), x_i>=0, x is a list of variables of length >= 1
def msk_geo_mean(task, t, x):
    n = len(x)
    if n==1:
        msk_abs(task, x[0], t)
    else:
        t2 = msk_newvar(task, 1)
        task.appendcone(mosek.conetype.ppow, 1.0-1.0/n, [t2, x[n-1], t])
        msk_geo_mean(task, msk_dup(task, t2), x[0:n-1])
```

6.9.4 Exponentials and logarithms

log

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

Listing 6.26: Logarithm.

```
# t <= log(x), x>=0
def msk_log(task, t, x):
    task.appendcone(mosek.conetype.pexp, 0.0, [x, msk_newvar_fx(task, 1, 1.0), t])
```

exp

$t \geq e^x$

Listing 6.27: Exponential.

```
# t >= exp(x)
def msk_exp(task, t, x):
    task.appendcone(mosek.conetype.pexp, 0.0, [t, msk_newvar_fx(task, 1, 1.0), x])
```

Entropy

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

Listing 6.28: Entropy.

```
# t >= x * log(x), x>=0
def msk_ent(task, t, x):
    v = msk_newvar(task, 1)
    c = msk_newcon(task, 1)
    task.putaij(c, v, 1.0)
    task.putaij(c, t, 1.0)
    task.putconbound(c, mosek.boundkey.fx, 0.0, 0.0)
    task.appendcone(mosek.conetype.pexp, 0.0, [msk_newvar_fx(task, 1, 1.0), x, v])
```

Relative entropy

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

Listing 6.29: Relative entropy.

```
# t >= x * log(x/y), x,y>=0
def msk_relent(task, t, x, y):
    v = msk_newvar(task, 1)
    c = msk_newcon(task, 1)
    task.putaij(c, v, 1.0)
    task.putaij(c, t, 1.0)
    task.putconbound(c, mosek.boundkey.fx, 0.0, 0.0)
    task.appendcone(mosek.conetype.pexp, 0.0, [y, x, v])
```

Log-sum-exp

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

Listing 6.30: Log-sum-exp.

```
# log( sum_i(exp(x_i)) ) <= t, where x is a list of variables
def msk_logsumexp(task, t, x):
    n = len(x)
    u = msk_newvar(task, n)
    z = msk_newvar(task, n)
    for i in range(n):
        msk_exp(task, u+i, z+i)
    c = msk_newcon(task, n)
    for i in range(n):
        task.putarow(c+i, [x[i], t, z+i], [1.0, -1.0, -1.0])
        task.putconbound(c+i, mosek.boundkey.fx, 0.0, 0.0)
    s = msk_newcon(task, 1)
    task.putarow(s, range(u, u+n), [1.0]*n)
    task.putconbound(s, mosek.boundkey.up, -inf, 1.0)
```

6.9.5 Integer Modeling

Semicontinuous variable

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

Listing 6.31: Semicontinuous variable.

```
# x = 0 or a <= x <= b
def msk_semicontinuous(task, x, a, b):
    u = msk_newvar_bin(task, 1)
    c = msk_newcon(task, 2)
    task.putarow(c, [x, u], [1.0, -a])
    task.putconbound(c, mosek.boundkey.lo, 0.0, inf)
    task.putarow(c+1, [x, u], [1.0, -b])
    task.putconbound(c+1, mosek.boundkey.up, -inf, 0.0)
```

Indicator variable

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

Listing 6.32: Indicator variable.

```
# x!=0 implies t=1. Assumes that |x|<=1 in advance.
def msk_indicator(task, x):
    t = msk_newvar_bin(task, 1)
    msk_abs(task, t, x)
    return t
```

Logical OR

At least one of the conditions is true.

Listing 6.33: Logical OR.

```
# x OR y, where x, y are binary
def msk_logic_or(task, x, y):
    c = msk_newcon(task, 1)
    task.putarow(c, [x, y], [1.0, 1.0])
    task.putconbound(c, mosek.boundkey.lo, 1.0, inf)
# x_1 OR ... OR x_n, where x is sequence of variables
def msk_logic_or_vect(task, x):
    c = msk_newcon(task, 1)
    n = len(x)
    task.putarow(c, x, [1.0]*n)
    task.putconbound(c, mosek.boundkey.lo, 1.0, inf)
```

Logical NAND

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

Listing 6.34: Logical NAND.

```
# at most one of x_1,...,x_n, where x is a binary vector (SOS1 constraint)
def msk_logic_sos1(task, x):
    c = msk_newcon(task, 1)
    n = len(x)
    task.putarow(c, x, [1.0]*n)
    task.putconbound(c, mosek.boundkey.up, -inf, 1.0)
# NOT(x AND y), where x, y are binary
def msk_logic_nand(task, x, y):
    c = msk_newcon(task, 1)
```

(continues on next page)

(continued from previous page)

```
    task.putarow(c, [x, y], [1.0, 1.0])
    task.putconbound(c, mosek.boundkey.up, -inf, 1.0)
```

Cardinality bound

At most k of the continuous variables are nonzero. We assume x is a priori normalized so $|x_i| \leq 1$.

Listing 6.35: Cardinality bound.

```
# At most k of entries in x are nonzero, assuming in advance that |x_i|<=1.
def msk_card(task, x, k):
    n = len(x)
    t = msk_newvar_bin(task, n)
    for i in range(n):
        msk_abs(task, t+i, x[i])
    c = msk_newcon(task, 1)
    task.putarow(c, range(t, t+n), [1.0]*n)
    task.putconbound(c, mosek.boundkey.up, -inf, k)
```

6.9.6 Model assembly example

We now demonstrate how to quickly build a simple optimization model for the problem

$$\begin{array}{ll} \text{maximize} & -\sqrt{x^2+y^2} + \log y - x^{1.5}, \\ \text{subject to} & x \geq y+3, \end{array} \tag{6.21}$$

or equivalently

$$\begin{array}{ll} \text{maximize} & -t_0 + t_1 - t_2, \\ \text{subject to} & x \geq y+3, \\ & t_0 \geq \sqrt{x^2+y^2}, \\ & t_1 \leq \log y, \\ & t_2 \geq x^{1.5}. \end{array}$$

Listing 6.36: Modeling (6.21).

```
def testExample():
    env = mosek.Env()
    task = env.Task()
    x = msk_newvar(task, 1)
    y = msk_newvar(task, 1)
    t = msk_newvar(task, 3)

    c = msk_newcon(task, 1)
    task.putarow(c, [x, y], [1.0, -1.0])
    task.putconbound(c, mosek.boundkey.lo, 3.0, inf)

    msk_norm2(task, t+0, [x,y])
    msk_log  (task, t+1, msk_dup(task, y))
    msk_pow  (task, t+2, msk_dup(task, x), 1.5)

    task.putclist(range(t, t+3), [-1.0, 1.0, -1.0])
    task.putobjsense(mosek.objsense.maximize)
```

6.10 Problem Modification and Reoptimization

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

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

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

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

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

In general, **MOSEK** exploits dual information and availability of an optimal basis from the previous execution. The simplex optimizer is well suited for exploiting an existing primal or dual feasible solution. Restarting capabilities for interior-point methods are still not as reliable and effective as those for the simplex algorithm. More information can be found in Chapter 10 of the book *[Chv83]*.

Parameter settings (see Sec. 7.4) can also be changed between optimizations.

6.10.1 Example: Production Planning

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

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

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

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

$$
\begin{array}{lccccccc}
\text{maximize} & 1.5x_0 & + & 2.5x_1 & + & 3.0x_2 & & \\
\text{subject to} & 2x_0 & + & 4x_1 & + & 3x_2 & \leq & 100000, \\
 & 3x_0 & + & 2x_1 & + & 3x_2 & \leq & 50000, \\
 & 2x_0 & + & 3x_1 & + & 2x_2 & \leq & 60000,
\end{array} \tag{6.22}
$$

and

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

Code in Listing 6.37 loads and solves this problem.

Listing 6.37: Setting up and solving problem (6.22)

```
# Create a MOSEK environment
with mosek.Env() as env:
    # Create a task
    with env.Task(0, 0) as task:
        # Bound keys for constraints
        bkc = [mosek.boundkey.up,
               mosek.boundkey.up,
               mosek.boundkey.up]
        # Bound values for constraints
```

(continues on next page)

(continued from previous page)

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

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

        # Append the constraints
        task.appendcons(numcon)

        # Append the variables.
        task.appendvars(numvar)

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

        # Put constraint bounds
        task.putconboundslice(0, numcon, bkc, blc, buc)

        # Put variable bounds
        task.putvarboundslice(0, numvar, bkx, blx, bux)

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

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

        # Optimize the task
        task.optimize()

        # Output a solution
        xx = [0.] * numvar
        task.getsolutionslice(mosek.soltype.bas,
```

(continues on next page)

(continued from previous page)

```
                                mosek.solitem.xx,
                                0, numvar,
                                xx)
            print("xx = {}".format(xx))
```

6.10.2 Changing the Linear Constraint Matrix

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

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

The problem now has the form:

$$
\begin{array}{lccccccc}
\text{maximize} & 1.5x_0 & + & 2.5x_1 & + & 3.0x_2 & & \\
\text{subject to} & 3x_0 & + & 4x_1 & + & 3x_2 & \leq & 100000, \\
 & 3x_0 & + & 2x_1 & + & 3x_2 & \leq & 50000, \\
 & 2x_0 & + & 3x_1 & + & 2x_2 & \leq & 60000,
\end{array} \tag{6.23}
$$

and

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

After this operation we can reoptimize the problem.

6.10.3 Appending Variables

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

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

This corresponds to creating a new variable x_3, appending a new column to the A matrix and setting a new term in the objective. We do this in Listing 6.38

Listing 6.38: How to add a new variable (column)

```
            ################## Add a new variable ######################
            task.appendvars(1)
            numvar+=1

            # Set bounds on new varaible
            task.putvarbound(task.getnumvar() - 1,
                             mosek.boundkey.lo,
                             0,
                             +inf)

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

            # Put new values in the A matrix
            acolsub = [0, 2]
            acolval = [4.0, 1.0]

            task.putacol(task.getnumvar() - 1, # column index
                         acolsub,
                         acolval)
```

After this operation the new problem is:

$$
\begin{array}{lccccccccc}
\text{maximize} & 1.5x_0 & + & 2.5x_1 & + & 3.0x_2 & + & 1.0x_3 & & \\
\text{subject to} & 3x_0 & + & 4x_1 & + & 3x_2 & + & 4x_3 & \leq & 100000, \\
 & 3x_0 & + & 2x_1 & + & 3x_2 & & & \leq & 50000, \\
 & 2x_0 & + & 3x_1 & + & 2x_2 & + & 1x_3 & \leq & 60000,
\end{array} \tag{6.24}
$$

and

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

6.10.4 Appending Constraints

Now suppose we want to add a new stage to the production process called *Quality control* for which 30000 minutes are available. The time requirement for this stage is shown below:

Product no.	Quality control (minutes)
0	1
1	2
2	1
3	1

This corresponds to adding the constraint

$$x_0 + 2x_1 + x_2 + x_3 \leq 30000$$

to the problem. This is done as follows.

Listing 6.39: Adding a new constraint.

```
############# Add a new constraint ######################
task.appendcons(1)
numcon+=1

# Set bounds on new constraint
task.putconbound(task.getnumcon() - 1,
                 mosek.boundkey.up, -inf, 30000)

# Put new values in the A matrix
arowsub = [0, 1, 2, 3]
arowval = [1.0, 2.0, 1.0, 1.0]

task.putarow(task.getnumcon() - 1, # row index
             arowsub,
             arowval)
```

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

6.10.5 Changing bounds

One typical reoptimization scenario is to change bounds. Suppose for instance that we must operate with limited time resources, and we must change the upper bounds in the problem as follows:

Operation	Time available (before)	Time available (new)
Assembly	100000	80000
Polishing	50000	40000
Packing	60000	50000
Quality control	30000	22000

That means we would like to solve the problem:

$$\begin{array}{lccccccccc}
\text{maximize} & 1.5x_0 & + & 2.5x_1 & + & 3.0x_2 & + & 1.0x_3 & & \\
\text{subject to} & 3x_0 & + & 4x_1 & + & 3x_2 & + & 4x_3 & \leq & 80000, \\
 & 3x_0 & + & 2x_1 & + & 3x_2 & & & \leq & 40000, \\
 & 2x_0 & + & 3x_1 & + & 2x_2 & + & 1x_3 & \leq & 50000, \\
 & x_0 & + & 2x_1 & + & x_2 & + & x_3 & \leq & 22000.
\end{array} \tag{6.25}$$

In this case all we need to do is redefine the upper bound vector for the constraints, as shown in the next listing.

Listing 6.40: Change constraint bounds.

```
############# Change constraint bounds ######################
newbkc = [mosek.boundkey.up] * numcon
newblc = [-inf] * numcon
newbuc = [ 80000, 40000, 50000, 22000 ]

task.putconboundslice(0, numcon, newbkc, newblc, newbuc)
```

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

6.10.6 Advanced hot-start

If the optimizer used the data from the previous run to hot-start the optimizer for reoptimization, this will be indicated in the log:

```
Optimizer  - hotstart                 : yes
```

When performing re-optimizations, instead of removing a basic variable it may be more efficient to fix the variable at zero and then remove it when the problem is re-optimized and it has left the basis. This makes it easier for **MOSEK** to restart the simplex optimizer.

6.11 Parallel optimization

In this section we demonstrate the simplest possible multi-threading setup to run multiple **MOSEK** optimizations in parallel. All tasks must be created using the same **MOSEK** environment. One license token checked out by the environment will be shared by the tasks.

We first define a simple method that runs a number of optimization tasks in parallel, using the standard multi-threading setup available in the language.

Listing 6.41: Parallel optimization of a list of tasks.

```
# A run method to optimize a single task
def runTask(num, task, res, trm):
  try:
    trm[num] = task.optimize();
    res[num] = mosek.rescode.ok
  except mosek.MosekException as e:
    trm[num] = mosek.rescode.err_unknown
    res[num] = e.errno

# Takes a list of tasks and optimizes them in parallel threads. The
# response code and termination code from each optimization is
#  stored in ``res`` and ``trm``.
def paropt(tasks):
  n = len(tasks)
  res = [ mosek.rescode.err_unknown ] * n
  trm = [ mosek.rescode.err_unknown ] * n

  # Start parallel optimizations, one per task
  jobs = [ Thread(target=runTask, args=(i, tasks[i], res, trm)) for i in range(n) ]
  for j in jobs:
    j.start()
  for j in jobs:
    j.join()

  return res, trm
```

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

Listing 6.42: Calling the parallel optimizer.

```python
# Example of how to use ``paropt``.
# Optimizes tasks whose names were read from command line.
def main(argv):
  n = len(argv) - 1
  tasks = []

  with mosek.Env() as env:
    for i in range(n):
      t = mosek.Task(env, 0, 0)
      t.readdata(argv[i+1])
      # Each task will be single-threaded
      t.putintparam(mosek.iparam.intpnt_multi_thread, mosek.onoffkey.off)
      tasks.append(t)

    res, trm = paropt(tasks)

    for i in range(n):
      print("Task  {0}  res {1}   trm {2}   obj_val  {3}  time {4}".format(
             i,
             res[i],
             trm[i],
             tasks[i].getdouinf(mosek.dinfitem.intpnt_primal_obj),
             tasks[i].getdouinf(mosek.dinfitem.optimizer_time)))
```

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

- *Case studies* for more advanced and complicated optimization examples.
- *Problem Formulation and Solutions* for formal mathematical formulations of problems **MOSEK** can solve, dual problems and infeasibility certificates.

Chapter 7

Solver Interaction Tutorials

In this section we cover the interaction with the solver.

7.1 Accessing the solution

This section contains important information about the status of the solver and the status of the solution, which must be checked in order to properly interpret the results of the optimization.

7.1.1 Solver termination

The optimizer provides two status codes relevant for error handling:

- **Response code** of type *rescode*. It indicates if any unexpected error (such as an out of memory error, licensing error etc.) has occurred. The expected value for a successful optimization is *rescode.ok*.
- **Termination code**: It provides information about why the optimizer terminated, for instance if a predefined time limit has been reached. These are not errors, but ordinary events that can be expected (depending on parameter settings and the type of optimizer used).

If the optimization was successful then the method *Task.optimize* returns normally and its output is the termination code. If an error occurs then the method throws an exception, which contains the response code. See Sec. 7.2 for how to access it.

If a runtime error causes the program to crash during optimization, the first debugging step is to enable logging and check the log output. See Sec. 7.3.

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

7.1.2 Available solutions

MOSEK uses three kinds of optimizers and provides three types of solutions:

- **basic solution** from the simplex optimizer,
- **interior-point solution** from the interior-point optimizer,
- **integer solution** from the mixed-integer optimizer.

Under standard parameters settings the following solutions will be available for various problem types:

Table 7.1: Types of solutions available from **MOSEK**

	Simplex optimizer	Interior-point optimizer	Mixed-integer optimizer
Linear problem	*soltype.bas*	*soltype.itr*	
Nonlinear continuous problem		*soltype.itr*	
Problem with integer variables			*soltype.itg*

For linear problems the user can force a specific optimizer choice making only one of the two solutions available. For example, if the user disables basis identification, then only the interior point solution will be available for a linear problem. Numerical issues may cause one of the solutions to be unknown even if another one is feasible.

Not all components of a solution are always available. For example, there is no dual solution for integer problems and no dual conic variables from the simplex optimizer.

The user will always need to specify which solution should be accessed.

7.1.3 Problem and solution status

Assuming that the optimization terminated without errors, the next important step is to check the problem and solution status. There is one for every type of solution, as explained above.

Problem status

Problem status (*prosta*) determines whether the problem is certified as feasible. Its values can roughly be divided into the following broad categories:

- **feasible** — the problem is feasible. For continuous problems and when the solver is run with default parameters, the feasibility status should ideally be *prosta.prim_and_dual_feas*.
- **primal/dual infeasible** — the problem is infeasible or unbounded or a combination of those. The exact problem status will indicate the type of infeasibility.
- **unknown** — the solver was unable to reach a conclusion, most likely due to numerical issues.

Solution status

Solution status (*solsta*) provides the information about what the solution values actually contain. The most important broad categories of values are:

- **optimal** (*solsta.optimal*) — the solution values are feasible and optimal.
- **certificate** — the solution is in fact a certificate of infeasibility (primal or dual, depending on the solution).
- **unknown/undefined** — the solver could not solve the problem or this type of solution is not available for a given problem.

Problem and solution status for each solution can be retrieved with *Task.getprosta* and *Task.getsolsta*, respectively.

The solution status determines the action to be taken. For example, in some cases a suboptimal solution may still be valuable and deserve attention. It is the user's responsibility to check the status and quality of the solution.

Typical status reports

Here are the most typical optimization outcomes described in terms of the problem and solution statuses. Note that these do not cover all possible situations that can occur.

Table 7.2: Continuous problems (solution status for interior-point and basic solution)

Outcome	Problem status	Solution status
Optimal	*prosta.prim_and_dual_feas*	*solsta.optimal*
Primal infeasible	*prosta.prim_infeas*	*solsta.prim_infeas_cer*
Dual infeasible (unbounded)	*prosta.dual_infeas*	*solsta.dual_infeas_cer*
Uncertain (stall, numerical issues, etc.)	*prosta.unknown*	*solsta.unknown*

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

Outcome	Problem status	Solution status
Integer optimal	*prosta.prim_feas*	*solsta.integer_optimal*
Infeasible	*prosta.prim_infeas*	*solsta.unknown*
Integer feasible point	*prosta.prim_feas*	*solsta.prim_feas*
No conclusion	*prosta.unknown*	*solsta.unknown*

7.1.4 Retrieving solution values

After the meaning and quality of the solution (or certificate) have been established, we can query for the actual numerical values. They can be accessed using:

- *Task.getprimalobj*, *Task.getdualobj* — the primal and dual objective value.
- *Task.getxx* — solution values for the variables.
- *Task.getsolution* — a full solution with primal and dual values

and many more specialized methods, see the *API reference*.

7.1.5 Source code example

Below is a source code example with a simple framework for assessing and retrieving the solution to a conic optimization problem.

Listing 7.1: Sample framework for checking optimization result.

```
import mosek
import sys

# A log message
def streamprinter(text):
    sys.stdout.write(text)
    sys.stdout.flush()

def main(args):
  filename = args[0] if len(args) >= 1 else "../data/cqo1.mps"

  try:
    # Create environment and task
    with mosek.Env() as env:
      with env.Task(0, 0) as task:
        # (Optional) set a log stream
        # task.set_Stream(mosek.streamtype.log, streamprinter)

        # (Optional) uncomment to see what happens when solution status is unknown
        # task.putintparam(mosek.iparam.intpnt_max_iterations, 1)

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

        # Optimize
        trmcode = task.optimize()
        task.solutionsummary(mosek.streamtype.log)

        # We expect solution status OPTIMAL
        solsta = task.getsolsta(mosek.soltype.itr)

        if solsta == mosek.solsta.optimal:
          # Optimal solution. Fetch and print it.
```

(continues on next page)

(continued from previous page)

```
        print("An optimal interior-point solution is located.")
        numvar = task.getnumvar()
        xx = [ 0.0 ] * numvar
        task.getxx(mosek.soltype.itr, xx)
        for i in range(numvar):
          print("x[{0}] = {1}".format(i, xx[i]))

      elif solsta == mosek.solsta.dual_infeas_cer:
        print("Dual infeasibility certificate found.")

      elif solsta == mosek.solsta.prim_infeas_cer:
        print("Primal infeasibility certificate found.")

      elif solsta == mosek.solsta.unknown:
        # The solutions status is unknown. The termination code
        # indicates why the optimizer terminated prematurely.
        print("The solution status is unknown.")
        symname, desc = mosek.Env.getcodedesc(trmcode)
        print("   Termination code: {0} {1}".format(symname, desc))

      else:
        print("An unexpected solution status {0} is obtained.".format(str(solsta)))

  except mosek.Error as e:
      print("Unexpected error ({0}) {1}".format(e.errno, e.msg))

if __name__ == '__main__':
    main(sys.argv[1:])
```

7.2 Errors and exceptions

Exceptions

Almost every function in Optimizer API for Python can throw an exception informing that the requested operation was not performed correctly, and indicating the type of error that occurred. This is the case in situations such as for instance:

- referencing a nonexisting variable (for example with too large index),
- defining an invalid value for a parameter,
- accessing an undefined solution,
- repeating a variable name, etc.

It is therefore a good idea to catch exceptions of type `Error`. The one case where it is *extremely important* to do so is when `Task.optimize` is invoked. We will say more about this in Sec. 7.1.

The exception contains a *response code* (element of the enum `rescode`) and short diagnostic messages. They can be accessed as in the following example.

```
try:
  task.putdouparam(mosek.dparam.intpnt_co_tol_rel_gap, -1.0e-7)
except mosek.Error as e:
  print("Response code {0}\nMessage       {1}".format(e.errno, e.msg))
```

It will produce as output:

```
Response code rescode.err_param_is_too_small
Message       The parameter value -1e-07 is too small for parameter 'MSK_DPAR_INTPNT_CO_TOL_
↪REL_GAP'.
```

Another way to obtain a human-readable string corresponding to a response code is the method `Env.getcodedesc`. A full list of exceptions, as well as response codes, can be found in the *API reference*.

Optimizer errors and warnings

The optimizer may also produce warning messages. They indicate non-critical but important events, that will not prevent solver execution, but may be an indication that something in the optimization problem might be improved. Warning messages are normally printed to a log stream (see Sec. 7.3). A typical warning is, for example:

```
MOSEK warning 53: A numerically large upper bound value  6.6e+09 is specified for constraint
↪'C69200' (46020).
```

Warnings can also be suppressed by setting the *iparam.max_num_warnings* parameter to zero, if they are well-understood.

Error and solution status handling example

Below is a source code example with a simple framework for handling major errors when assessing and retrieving the solution to a conic optimization problem.

Listing 7.2: Sample framework for checking optimization result.

```
import mosek
import sys

# A log message
def streamprinter(text):
    sys.stdout.write(text)
    sys.stdout.flush()

def main(args):
  filename = args[0] if len(args) >= 1 else "../data/cqo1.mps"

  try:
    # Create environment and task
    with mosek.Env() as env:
      with env.Task(0, 0) as task:
        # (Optional) set a log stream
        # task.set_Stream(mosek.streamtype.log, streamprinter)

        # (Optional) uncomment to see what happens when solution status is unknown
        # task.putintparam(mosek.iparam.intpnt_max_iterations, 1)

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

        # Optimize
        trmcode = task.optimize()
        task.solutionsummary(mosek.streamtype.log)

        # We expect solution status OPTIMAL
        solsta = task.getsolsta(mosek.soltype.itr)

        if solsta == mosek.solsta.optimal:
          # Optimal solution. Fetch and print it.
          print("An optimal interior-point solution is located.")
          numvar = task.getnumvar()
          xx = [ 0.0 ] * numvar
          task.getxx(mosek.soltype.itr, xx)
          for i in range(numvar):
            print("x[{0}] = {1}".format(i, xx[i]))

        elif solsta == mosek.solsta.dual_infeas_cer:
          print("Dual infeasibility certificate found.")
```

(continues on next page)

(continued from previous page)

```
        elif solsta == mosek.solsta.prim_infeas_cer:
          print("Primal infeasibility certificate found.")

        elif solsta == mosek.solsta.unknown:
          # The solutions status is unknown. The termination code
          # indicates why the optimizer terminated prematurely.
          print("The solution status is unknown.")
          symname, desc = mosek.Env.getcodedesc(trmcode)
          print("   Termination code: {0} {1}".format(symname, desc))

        else:
          print("An unexpected solution status {0} is obtained.".format(str(solsta)))

  except mosek.Error as e:
      print("Unexpected error ({0}) {1}".format(e.errno, e.msg))

if __name__ == '__main__':
    main(sys.argv[1:])
```

7.3 Input/Output

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

7.3.1 Stream logging

By default the solver runs silently and does not produce any output to the console or otherwise. However, the log output can be redirected to a user-defined output stream or stream callback function. The log output is analogous to the one produced by the command-line version of **MOSEK**.

The log messages are partitioned in three streams:

- messages, *streamtype.msg*
- warnings, *streamtype.wrn*
- errors, *streamtype.err*

These streams are aggregated in the *streamtype.log* stream. A stream handler can be defined for each stream separately.

A stream handler is simply a user-defined function of type *streamfunc* that accepts a string, for example:

```
def myStream(msg):
    sys.stdout.write(msg)
    sys.stdout.flush()
```

It is attached to a stream as follows:

```
task.set_Stream(streamtype.log,myStream)
```

The stream can be detached by calling

```
task.set_Stream(streamtype.log,None)
```

After optimization is completed an additional short summary of the solution and optimization process can be printed to any stream using the method *Task.solutionsummary*.

7.3.2 Log verbosity

The logging verbosity can be controlled by setting the relevant parameters, as for instance

- *iparam.log*,
- *iparam.log_intpnt*,
- *iparam.log_mio*,
- *iparam.log_cut_second_opt*,
- *iparam.log_sim*, and
- *iparam.log_sim_minor*.

Each parameter controls the output level of a specific functionality or algorithm. The main switch is *iparam.log* which affect the whole output. The actual log level for a specific functionality is determined as the minimum between *iparam.log* and the relevant parameter. For instance, the log level for the output produce by the interior-point algorithm is tuned by the *iparam.log_intpnt*; the actual log level is defined by the minimum between *iparam.log* and *iparam.log_intpnt*.

Tuning the solver verbosity may require adjusting several parameters. It must be noticed that verbose logging is supposed to be of interest during debugging and tuning. When output is no more of interest, the user can easily disable it globally with *iparam.log*. Larger values of *iparam.log* do not necessarily result in increased output.

By default **MOSEK** will reduce the amount of log information after the first optimization on a given problem. To get full log output on subsequent re-optimizations set *iparam.log_cut_second_opt* to zero.

7.3.3 Saving a problem to a file

An optimization problem can be dumped to a file using the method *Task.writedata*. The file format will be determined from the extension of the filename. Supported formats are listed in Sec. 16 together with a table of problem types supported by each.

For instance the problem can be written to an OPF file with

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

All formats can be compressed with `gzip` by appending the `.gz` extension, for example

```
task.writedata("data.task.gz")
```

Some remarks:

- Unnamed variables are given generic names. It is therefore recommended to use meaningful variable names if the problem file is meant to be human-readable.
- The `task` format is **MOSEK**'s native file format which contains all the problem data as well as solver settings.

7.3.4 Reading a problem from a file

A problem saved in any of the supported file formats can be read directly into a task using *Task.readdata*. The task must be created in advance. Afterwards the problem can be optimized, modified, etc. If the file contained solutions, then are also imported, but the status of any solution will be set to *solsta.unknown* (solutions can also be read separately using *Task.readsolution*). If the file contains parameters, they will be set accordingly.

```
task = env.Task()
try:
    task.readdata("file.task.gz")
    task.optimize()
except mosek.Error:
    print("Problem reading the file")
```

7.4 Setting solver parameters

MOSEK comes with a large number of parameters that allows the user to tune the behavior of the optimizer. The typical settings which can be changed with solver parameters include:

- choice of the optimizer for linear problems,
- choice of primal/dual solver,
- turning presolve on/off,
- turning heuristics in the mixed-integer optimizer on/off,
- level of multi-threading,
- feasibility tolerances,
- solver termination criteria,
- behaviour of the license manager,

and more. All parameters have default settings which will be suitable for most typical users.
The API reference contains:

- *Full list of parameters*
- *List of parameters grouped by topic*

Setting parameters

Each parameter is identified by a unique name. There are three types of parameters depending on the values they take:

- Integer parameters. They take either either simple integer values or values from an enumeration provided for readability and compatibility of the code. Set with *Task.putintparam*.
- Double (floating point) parameters. Set with *Task.putdouparam*.
- String parameters. Set with *Task.putstrparam*.

There are also parameter setting functions which operate fully on symbolic strings containing generic command-line style names of parameters and their values. See the example below. The optimizer will try to convert the given argument to the exact expected type, and will error if that fails.

If an incorrect value is provided then the parameter is left unchanged.

For example, the following piece of code sets up parameters which choose and tune the interior point optimizer before solving a problem.

Listing 7.3: Parameter setting example.

```
        # Set log level (integer parameter)
        task.putintparam(mosek.iparam.log, 1)
        # Select interior-point optimizer... (integer parameter)
        task.putintparam(mosek.iparam.optimizer, mosek.optimizertype.intpnt)
        # ... without basis identification (integer parameter)
        task.putintparam(mosek.iparam.intpnt_basis, mosek.basindtype.never)
        # Set relative gap tolerance (double parameter)
        task.putdouparam(mosek.dparam.intpnt_co_tol_rel_gap, 1.0e-7)

        # The same using explicit string names
        task.putparam     ("MSK_DPAR_INTPNT_CO_TOL_REL_GAP", "1.0e-7")
        task.putnadouparam("MSK_DPAR_INTPNT_CO_TOL_REL_GAP",  1.0e-7 )

        # Incorrect value
        try:
            task.putdouparam(mosek.dparam.intpnt_co_tol_rel_gap, -1.0)
        except:
            print('Wrong parameter value')
```

Reading parameter values

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

```
param = task.getdouparam(mosek.dparam.intpnt_co_tol_rel_gap)
print('Current value for parameter intpnt_co_tol_rel_gap = {}'.format(param))
```

7.5 Retrieving information items

After the optimization the user has access to the solution as well as to a report containing a large amount of additional *information items*. For example, one can obtain information about:

- **timing**: total optimization time, time spent in various optimizer subroutines, number of iterations, etc.
- **solution quality**: feasibility measures, solution norms, constraint and bound violations, etc.
- **problem structure**: counts of variables of different types, constraints, nonzeros, etc.
- **integer optimizer**: integrality gap, objective bound, number of cuts, etc.

and more. Information items are numerical values of integer, long integer or double type. The full list can be found in the API reference:

- *Double*
- *Integer*
- *Long*

Certain information items make sense, and are made available, also *during* the optimization process. They can be accessed from a callback function, see Sec. 7.6 for details.

Remark

For efficiency reasons, not all information items are automatically computed after optimization. To force all information items to be updated use the parameter *iparam.auto_update_sol_info*.

Retrieving the values

Values of information items are fetched using one of the methods

- *Task.getdouinf* for a double information item,
- *Task.getintinf* for an integer information item,
- *Task.getlintinf* for a long integer information item.

Each information item is identified by a unique name. The example below reads two pieces of data from the solver: total optimization time and the number of interior-point iterations.

Listing 7.4: Information items example.

```
tm = task.getdouinf(mosek.dinfitem.optimizer_time)
it = task.getintinf(mosek.iinfitem.intpnt_iter)

print('Time: {0}\nIterations: {1}'.format(tm,it))
```

7.6 Progress and data callback

Callbacks are a very useful mechanism that allow the caller to track the progress of the **MOSEK** optimizer. A callback function provided by the user is regularly called during the optimization and can be used to

- obtain a customized log of the solver execution,
- collect information for debugging purposes or
- ask the solver to terminate.

Optimizer API for Python has the following callback mechanisms:

- **progress callback**, which provides only the basic status of the solver.
- **data callback**, which provides the solver status and a complete set of information items that describe the progress of the optimizer in detail.

Warning

The callbacks functions *must not* invoke any functions of the solver, environment or task. Otherwise the state of the solver and its outcome are undefined. The only exception is the possibility to retrieve an integer solution, see below.

Retrieving mixed-integer solutions

If the mixed-integer optimizer is used, the callback will take place, in particular, every time an improved integer solution is found. In that case it is possible to retrieve the current values of the best integer solution from within the callback function. It can be useful for implementing complex termination criteria for integer optimization. The example in Listing 7.5 shows how to do it by handling the callback code `callbackcode.new_int_mio`.

7.6.1 Data callback

In the data callback **MOSEK** passes a callback code and values of all information items to a user-defined function. The callback function is called, in particular, at the beginning of each iteration of the interior-point optimizer. For the simplex optimizers `iparam.log_sim_freq` controls how frequently the call-back is called. Note that the callback is done quite frequently, which can lead to degraded performance. If the information items are not required, the simpler progress callback may be a better choice.

The callback is set by calling the method `Task.set_InfoCallback` and providing a handle to a user-defined function `callbackfunc`.

Non-zero return value of the callback function indicates that the optimizer should be terminated.

7.6.2 Progress callback

In the progress callback **MOSEK** provides a single code indicating the current stage of the optimization process.

The callback is set by calling the method `Task.set_Progress` and providing a handle to a user-defined function `progresscallbackfunc`.

Non-zero return value of the callback function indicates that the optimizer should be terminated.

7.6.3 Working example: Data callback

The following example defines a data callback function that prints out some of the information items. It interrupts the solver after a certain time limit.

Listing 7.5: An example of a data callback function.

```
def makeUserCallback(maxtime, task):
    xx = numpy.zeros(task.getnumvar())    # Space for integer solutions

    def userCallback(caller,
                     douinf,
                     intinf,
                     lintinf):
        opttime = 0.0

        if caller == callbackcode.begin_intpnt:
            print("Starting interior-point optimizer")
        elif caller == callbackcode.intpnt:
            itrn = intinf[iinfitem.intpnt_iter]
            pobj = douinf[dinfitem.intpnt_primal_obj]
            dobj = douinf[dinfitem.intpnt_dual_obj]
            stime = douinf[dinfitem.intpnt_time]
            opttime = douinf[dinfitem.optimizer_time]

            print("Iterations: %-3d" % itrn)
            print("  Elapsed time: %6.2f(%.2f) " % (opttime, stime))
            print("  Primal obj.: %-18.6e  Dual obj.: %-18.6e" % (pobj, dobj))
        elif caller == callbackcode.end_intpnt:
            print("Interior-point optimizer finished.")
        elif caller == callbackcode.begin_primal_simplex:
            print("Primal simplex optimizer started.")
        elif caller == callbackcode.update_primal_simplex:
            itrn = intinf[iinfitem.sim_primal_iter]
            pobj = douinf[dinfitem.sim_obj]
            stime = douinf[dinfitem.sim_time]
            opttime = douinf[dinfitem.optimizer_time]

            print("Iterations: %-3d" % itrn)
            print("  Elapsed time: %6.2f(%.2f)" % (opttime, stime))
            print("  Obj.: %-18.6e" % pobj)
        elif caller == callbackcode.end_primal_simplex:
            print("Primal simplex optimizer finished.")
        elif caller == callbackcode.begin_dual_simplex:
            print("Dual simplex optimizer started.")
        elif caller == callbackcode.update_dual_simplex:
            itrn = intinf[iinfitem.sim_dual_iter]
            pobj = douinf[dinfitem.sim_obj]
            stime = douinf[dinfitem.sim_time]
            opttime = douinf[dinfitem.optimizer_time]
            print("Iterations: %-3d" % itrn)
            print("  Elapsed time: %6.2f(%.2f)" % (opttime, stime))
            print("  Obj.: %-18.6e" % pobj)
        elif caller == callbackcode.end_dual_simplex:
            print("Dual simplex optimizer finished.")
        elif caller == callbackcode.new_int_mio:
            print("New integer solution has been located.")
            task.getxx(soltype.itg, xx)
            print(xx)
            print("Obj.: %f" % douinf[dinfitem.mio_obj_int])
        else:
            pass

        if opttime >= maxtime:
            # mosek is spending too much time. Terminate it.
            print("Terminating.")
            return 1
```

(continues on next page)

(continued from previous page)

```
        return 0
    return userCallback
```

Assuming that we have defined a task `task` and a time limit `maxtime`, the callback function is attached as follows:

Listing 7.6: Attaching the data callback function to the model.

```
            usercallback = makeUserCallback(maxtime=0.05, task=task)
            task.set_InfoCallback(usercallback)
```

7.7 MOSEK OptServer

MOSEK provides an easy way to offload optimization problem to a remote server. This section demonstrates related functionalities from the client side, i.e. sending optimization tasks to the remote server and retrieving solutions.

Setting up and configuring the remote server is described in a separate manual for the OptServer.

7.7.1 Synchronous Remote Optimization

In synchronous mode the client sends an optimization problem to the server and blocks, waiting for the optimization to end. Once the result has been received, the program can continue. This is the simplest mode all it takes is to provide the host and port where the server is running and listening as additional arguments. The rest of the code remains untouched.

Note that it is impossible to recover the job in case of a broken connection.

Source code example

Listing 7.7: Using the OptServer in synchronous mode.

```
import mosek
import sys

def streamprinter(msg):
    sys.stdout.write(msg)
    sys.stdout.flush()

if len(sys.argv) <= 3:
    print("Missing argument, syntax is:")
    print("  opt_server_sync inputfile host port")
else:

    inputfile = sys.argv[1]
    host = sys.argv[2]
    port = sys.argv[3]

    # Create the mosek environment.
    with mosek.Env() as env:

        # Create a task object linked with the environment env.
        # We create it with 0 variables and 0 constraints initially,
        # since we do not know the size of the problem.
        with env.Task(0, 0) as task:
            task.set_Stream(mosek.streamtype.log, streamprinter)

            # We assume that a problem file was given as the first command
            # line argument (received in `argv')
```

(continues on next page)

(continued from previous page)

```
        task.readdata(inputfile)

        # Solve the problem remotely
        task.optimizermt(host, port)

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

7.7.2 Asynchronous Remote Optimization

In asynchronous mode the client sends a job to the remote server and the execution of the client code continues. In particular, it is the client's responsibility to periodically check the optimization status and, when ready, fetch the results. The client can also interrupt optimization. The most relevant methods are:

- *Task.asyncoptimize* : Offload the optimization task to a solver server.
- *Task.asyncpoll* : Request information about the status of the remote job.
- *Task.asyncgetresult* : Request the results from a completed remote job.
- *Task.asyncstop* : Terminate a remote job.

Source code example

In the example below the program enters in a polling loop that regularly checks whether the result of the optimization is available.

Listing 7.8: Using the OptServer in asynchronous mode.

```
import mosek
import sys
import time

def streamprinter(msg):
    sys.stdout.write(msg)
    sys.stdout.flush()

if len(sys.argv) != 5:
    print("Missing argument, syntax is:")
    print("  opt-server-async inputfile host port numpolls")
else:

    filename = sys.argv[1]
    host = sys.argv[2]
    port = sys.argv[3]
    numpolls = int(sys.argv[4])
    token = None

    with mosek.Env() as env:

        with env.Task(0, 0) as task:

            print("reading task from file")
            task.readdata(filename)

            print("Solve the problem remotely (async)")
            token = task.asyncoptimize(host, port)

        print("Task token: %s" % token)
```

(continues on next page)

(continued from previous page)

```
        with env.Task(0, 0) as task:

            task.readdata(filename)

            task.set_Stream(mosek.streamtype.log, streamprinter)

            i = 0

            while i < numpolls:

                time.sleep(0.1)

                print("poll %d..." % i)
                respavailable, res, trm = task.asyncpoll(host,
                                                         port,
                                                         token)

                print("done!")

                if respavailable:
                    print("solution available!")

                    respavailable, res, trm = task.asyncgetresult(host,
                                                                  port,
                                                                  token)

                    task.solutionsummary(mosek.streamtype.log)
                    break

                i = i + 1

                if i == numpolls:
                    print("max number of polls reached, stopping host.")
                    task.asyncstop(host, port, token)
```

Chapter 8

Debugging Tutorials

This collection of tutorials contains basic techniques for debugging optimization problems using tools available in **MOSEK**: optimizer log, solution summary, infeasibility report, command-line tools. It is intended as a first line of technical help for issues such as: Why do I get solution status *unknown* and how can I fix it? Why is my model infeasible while it shouldn't be? Should I change some parameters? Can the model solve faster? etc.

The major steps when debugging a model are always:

- Enable log output. See Sec. 7.3.1 for how to do it. In the simplest case:

  Create a log handler function:

  ```
  def myStream(msg):
      sys.stdout.write(msg)
      sys.stdout.flush()
  ```

  attach it to the log stream:

  ```
  task.set_Stream(streamtype.log,myStream)
  ```

  and include solution summary after the call to `optimize`:

  ```
  task.optimize()
  task.solutionsummary(streamtype.log)
  ```

- Run the optimization and analyze the log output, see Sec. 8.1. In particular:
  - check if the problem setup (number of constraints/variables etc.) matches your expectation.
  - check solution summary and solution status.
- Dump the problem to disk if necessary to continue analysis. See Sec. 7.3.3.
  - use a human-readable text format, such as `*.opf` if you want to check the problem structure by hand. Assign names to variables and constraints to make them easier to identify.

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

  - use the **MOSEK** native format `*.task.gz` when submitting a bug report or support question.

    ```
    task.writedata("data.task.gz")
    ```

- Fix problem setup, improve the model, locate infeasibility or adjust parameters, depending on the diagnosis.

See the following sections for details.

8.1 Understanding optimizer log

The optimizer produces a log which splits roughly into four sections:

1. summary of the input data,
2. presolve and other pre-optimize problem setup stages,
3. actual optimizer iterations,
4. solution summary.

In this tutorial we show how to analyze the most important parts of the log when initially debugging a model: input data (1) and solution summary (4). For the iterations log (3) see Sec. 13.3.4 or Sec. 13.4.8.

8.1.1 Input data

If **MOSEK** behaves very far from expectations it may be due to errors in problem setup. The log file will begin with a summary of the structure of the problem, which looks for instance like:

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

This can be consulted to eliminate simple errors: wrong objective sense, wrong number of variables etc. Note that Fusion, and third-party modeling tools can introduce additional variables and constraints to the model. In the remaining **MOSEK** APIs the problem dimensions should match exactly what the user specified.

If this is not sufficient a bit more information can be obtained by dumping the problem to a file (see Sec. 8) and using the `anapro` option of any of the command line tools. It can also be done directly with the function *`Task.analyzeproblem`*. This will produce a longer summary similar to:

```
** Variables
scalar: 20414     integer: 0        matrix: 0
low: 2082         up: 5014          ranged: 0         free: 12892       fixed: 426

** Constraints
all: 20413
low: 10028        up: 0             ranged: 0         free: 0           fixed: 10385

** Cones
QUAD: 1           dims: 2865: 1
RQUAD: 2507       dims: 3: 2507

** Problem data (numerics)
|c|               nnz: 10028        min=2.09e-05      max=1.00e+00
|A|               nnz: 597023       min=1.17e-10      max=1.00e+00
blx               fin: 2508         min=-3.60e+09     max=2.75e+05
bux               fin: 5440         min=0.00e+00      max=2.94e+08
blc               fin: 20413        min=-7.61e+05     max=7.61e+05
buc               fin: 10385        min=-5.00e-01     max=0.00e+00
```

Again, this can be used to detect simple errors, such as:

- Wrong type of cone was used or it has wrong dimension.
- The bounds for variables or constraints are incorrect or incomplete. Check if you defined bound keys for all variables. A variable for which no bound was defined is by default fixed at 0.

- The model is otherwise incomplete.
- Suspicious values of coefficients.
- For various data sizes the model does not scale as expected.

Finally saving the problem in a human-friendly text format such as LP or OPF (see Sec. 8) and analyzing it by hand can reveal if the model is correct.

Warnings and errors

At this stage the user can encounter warnings which should not be ignored, unless they are well-understood. They can also serve as hints as to numerical issues with the problem data. A typical warning of this kind is

```
MOSEK warning 53: A numerically large upper bound value  2.9e+08 is specified for variable
↪'absh[107]' (2613).
```

Warnings do not stop the problem setup. If, on the other hand, an error occurs then the model will become invalid. The user should make sure to test for errors/exceptions from all API calls that set up the problem and validate the data. See Sec. 7.2 for more details.

8.1.2 Solution summary

The last item in the log is the solution summary. In the Optimizer API it is only printed by invoking the function *Task.solutionsummary*.

Continuous problem

Optimal solution

A typical solution summary for a continuous (linear, conic, quadratic) problem looks like:

```
Problem status  : PRIMAL_AND_DUAL_FEASIBLE
Solution status : OPTIMAL
Primal.  obj: 8.7560516107e+01    nrm: 1e+02    Viol.  con: 3e-12    var: 0e+00    cones: 3e-11
Dual.    obj: 8.7560521345e+01    nrm: 1e+00    Viol.  con: 5e-09    var: 9e-11    cones: 0e+00
```

It contains the following elements:

- Problem and solution status. For details see Sec. 7.1.3.
- A summary of the primal solution: objective value, infinity norm of the solution vector `xx`, maximal violations of constraints, variable bounds and cones. The violation of a linear constraint such as $a^T x \leq b$ is $\max(a^T x - b, 0)$. The violation of a conic constraint $x \in \mathcal{K}$ is the distance $\mathrm{dist}(x, \mathcal{K})$.
- The same for the dual solution.

The features of the solution summary which characterize a very good and accurate solution and a well-posed model are:

- **Status:** The solution status is `OPTIMAL`.
- **Duality gap:** The primal and dual objective values are (almost) identical, which proves the solution is (almost) optimal.
- **Norms:** Ideally the norms of the solution and the objective values should not be too large. This of course depends on the input data, but a huge solution norm can be an indicator of issues with the scaling, conditioning and/or well-posedness of the model. It may also indicate that the problem is borderline between feasibility and infeasibility and sensitive to small perturbations in this respect.
- **Violations:** The violations are close to zero, which proves the solution is (almost) feasible. Observe that due to rounding errors it can be expected that the violations are proportional to the norm (`nrm:`) of the solution. It is rarely the case that violations are exactly zero.

Solution status UNKNOWN

A typical example with solution status UNKNOWN due to numerical problems will look like:

```
Problem status  : UNKNOWN
Solution status : UNKNOWN
Primal.  obj: 1.3821656824e+01    nrm: 1e+01    Viol.  con: 2e-03    var: 0e+00    cones: 0e+00
Dual.    obj: 3.0119004098e-01    nrm: 5e+07    Viol.  con: 4e-16    var: 1e-01    cones: 0e+00
```

Note that:

- The primal and dual objective are very different.
- The dual solution has very large norm.
- There are considerable violations so the solution is likely far from feasible.

Follow the hints in Sec. 8.2 to resolve the issue.

Solution status UNKNOWN with a potentially useful solution

Solution status UNKNOWN does not necessarily mean that the solution is completely useless. It only means that the solver was unable to make any more progress due to numerical difficulties, and it was not able to reach the accuracy required by the termination criteria (see Sec. 13.3.2). Consider for instance:

```
Problem status  : UNKNOWN
Solution status : UNKNOWN
Primal.  obj: 3.4531019648e+04    nrm: 1e+05    Viol.  con: 7e-02    var: 0e+00    cones: 0e+00
Dual.    obj: 3.4529720645e+04    nrm: 8e+03    Viol.  con: 1e-04    var: 2e-04    cones: 0e+00
```

Such a solution may still be useful, and it is always up to the user to decide. It may be a good enough approximation of the optimal point. For example, the large constraint violation may be due to the fact that one constraint contained a huge coefficient.

Infeasibility certificate

A primal infeasibility certificate is stored in the dual variables:

```
Problem status  : PRIMAL_INFEASIBLE
Solution status : PRIMAL_INFEASIBLE_CER
Dual.    obj: 2.9238975853e+02    nrm: 6e+02    Viol.  con: 0e+00    var: 1e-11    cones: 0e+00
```

It is a Farkas-type certificate as described in Sec. 12.2.2. In particular, for a good certificate:

- The dual objective is positive for a minimization problem, negative for a maximization problem. Ideally it is well bounded away from zero.
- The norm is not too big and the violations are small (as for a solution).

If the model was not expected to be infeasible, the likely cause is an error in the problem formulation. Use the hints in Sec. 8.1.1 and Sec. 8.3 to locate the issue.

Just like a solution, the infeasibility certificate can be of better or worse quality. The infeasibility certificate above is very solid. However, there can be less clear-cut cases, such as for example:

```
Problem status  : PRIMAL_INFEASIBLE
Solution status : PRIMAL_INFEASIBLE_CER
Dual.    obj: 1.6378689238e-06    nrm: 6e+05    Viol.  con: 7e-03    var: 2e-04    cones: 0e+00
```

This infeasibility certificate is more dubious because the dual objective is positive, but barely so in comparison with the large violations. It also has rather large norm. This is more likely an indication that the problem is borderline between feasibility and infeasibility or simply ill-posed and sensitive to tiny variations in input data. See Sec. 8.3 and Sec. 8.2.

The same remarks apply to dual infeasibility (i.e. unboundedness) certificates. Here the primal objective should be negative a minimization problem and positive for a maximization problem.

8.1.3 Mixed-integer problem

Optimal integer solution

For a mixed-integer problem there is no dual solution and a typical optimal solution report will look as follows:

```
Problem status  : PRIMAL_FEASIBLE
Solution status : INTEGER_OPTIMAL
Primal.  obj: 6.0111122960e+06    nrm: 1e+03    Viol.  con: 2e-13    var: 2e-14    itg: 5e-15
```

The interpretation of all elements is as for a continuous problem. The additional field `itg` denotes the maximum violation of an integer variable from being an exact integer.

Feasible integer solution

If the solver found an integer solution but did not prove optimality, for instance because of a time limit, the solution status will be `PRIMAL_FEASIBLE`:

```
Problem status  : PRIMAL_FEASIBLE
Solution status : PRIMAL_FEASIBLE
Primal.  obj: 6.0114607792e+06    nrm: 1e+03    Viol.  con: 2e-13    var: 2e-13    itg: 4e-15
```

In this case it is valuable to go back to the optimizer summary to see how good the best solution is:

```
31       35       1        0        6.0114607792e+06     6.0078960892e+06     0.06        4.1

Objective of best integer solution : 6.011460779193e+06
Best objective bound               : 6.007896089225e+06
```

In this case the best integer solution found has objective value `6.011460779193e+06`, the best proved lower bound is `6.007896089225e+06` and so the solution is guaranteed to be within 0.06% from optimum. The same data can be obtained as information items through an API. See also Sec. 13.4 for more details.

Infeasible problem

If the problem is declared infeasible the summary is simply

```
Problem status  : PRIMAL_INFEASIBLE
Solution status : UNKNOWN
Primal.  obj: 0.0000000000e+00    nrm: 0e+00    Viol.  con: 0e+00    var: 0e+00    itg: 0e+00
```

If infeasibility was not expected, consult Sec. 8.3.

8.2 Addressing numerical issues

The suggestions in this section should help diagnose and solve issues with numerical instability, in particular `UNKNOWN` solution status or solutions with large violations. Since numerically stable models tend to solve faster, following these hints can also dramatically shorten solution times.

We always recommend that issues of this kind are addressed by reformulating or rescaling the model, since it is the modeler who has the best insight into the structure of the problem and can fix the cause of the issue.

8.2.1 Formulating problems

Scaling

Make sure that all the data in the problem are of comparable orders of magnitude. This applies especially to the linear constraint matrix. Use Sec. 8.1.1 if necessary. For example a report such as

```
|A|               nnz: 597023       min=1.17e-6       max=2.21e+5
```

means that the ratio of largest to smallest elements in `A` is 10^{11}. In this case the user should rescale or reformulate the model to avoid such spread which makes it difficult for **MOSEK** to scale the problem internally. In many cases it may be possible to change the units, i.e. express the model in terms of rescaled variables (for instance work with millions of dollars instead of dollars, etc.).

Similarly, if the objective contains very different coefficients, say

$$\text{maximize } 10^{10}x + y$$

then it is likely to lead to inaccuracies. The objective will be dominated by the contribution from x and y will become insignificant.

Removing huge bounds

Never use a very large number as replacement for ∞. Instead define the variable or constraint as unbounded from below/above. Similarly, avoid artificial huge bounds if you expect they will not become tight in the optimal solution.

Avoiding linear dependencies

As much as possible try to avoid linear dependencies and near-linear dependencies in the model. See Example 8.3.

Avoiding ill-posedness

Avoid continuous models which are ill-posed: the solution space is degenerate, for example consists of a single point (technically, the Slater condition is not satisfied). In general, this refers to problems which are borderline between feasible and infeasible. See Example 8.1.

Scaling the expected solution

Try to formulate the problem in such a way that the expected solution (both primal and dual) is not very large. Consult the solution summary Sec. 8.1.2 to check the objective values or solution norms.

8.2.2 Further suggestions

Here are other simple suggestions that can help locate the cause of the issues. They can also be used as hints for how to tune the optimizer if fixing the root causes of the issue is not possible.

- Remove the objective and solve the feasibility problem. This can reveal issues with the objective.
- Change the objective or change the objective sense from minimization to maximization (if applicable). If the two objective values are almost identical, this may indicate that the feasible set is very small, possibly degenerate.
- Perturb the data, for instance bounds, very slightly, and compare the results.
- For linear problems: solve the problem using a different optimizer by setting the parameter *iparam.optimizer* and compare the results.
- Force the optimizer to solve the primal/dual versions of the problem by setting the parameter *iparam.intpnt_solve_form* or *iparam.sim_solve_form*. **MOSEK** has a heuristic to decide whether to dualize, but for some problems the guess is wrong an explicit choice may give better results.
- Solve the problem without presolve or some of its parts by setting the parameter *iparam.presolve_use*, see Sec. 13.1.
- Use different numbers of threads (*iparam.num_threads*) and compare the results. Very different results indicate numerical issues resulting from round-off errors.

If the problem was dumped to a file, experimenting with various parameters is facilitated with the **MOSEK** Command Line Tool or **MOSEK** Python Console Sec. 8.4.

8.2.3 Typical pitfalls

Example 8.1 (Ill-posedness). A toy example of this situation is the feasibility problem

$$(x-1)^2 \leq 1, \ (x+1)^2 \leq 1$$

whose only solution is $x = 0$ and moreover replacing any 1 on the right hand side by $1 - \varepsilon$ makes the problem infeasible and replacing it by $1 + \varepsilon$ yields a problem whose solution set is an interval (fully-dimensional). This is an example of ill-posedness.

Example 8.2 (Huge solution). If the norm of the expected solution is very large it may lead to numerical issues or infeasibility. For example the problem

$$(10^{-4}, x, 10^3) \in \mathcal{Q}_r^3$$

may be declared infeasible because the expected solution must satisfy $x \geq 5 \cdot 10^9$.

Example 8.3 (Near linear dependency). Consider the following problem:

$$\begin{array}{lrrrrrrrrrl}
\text{minimize} & & & & & & & & & & \\
\text{subject to} & & x_1 & + & x_2 & & & & & = & 1, \\
 & & & & & & x_3 & + & x_4 & = & 1, \\
 & - & x_1 & & & - & x_3 & & & = & -1+\varepsilon, \\
 & & & - & x_2 & & & - & x_4 & = & -1, \\
 & & x_1, & & x_2, & & x_3, & & x_4 & \geq & 0.
\end{array}$$

If we add the equalities together we obtain:

$$0 = \varepsilon$$

which is infeasible for any $\varepsilon \neq 0$. Here infeasibility is caused by a linear dependency in the constraint matrix coupled with a precision error represented by the ε. Indeed if a problem contains linear dependencies then the problem is either infeasible or contains redundant constraints. In the above case any of the equality constraints can be removed while not changing the set of feasible solutions. To summarize linear dependencies in the constraints can give rise to infeasible problems and therefore it is better to avoid them.

Example 8.4 (Presolving very tight bounds). Next consider the problem

$$\begin{array}{lccc}
\text{minimize} & & & \\
\text{subject to} & x_1 - 0.01x_2 & = & 0, \\
 & x_2 - 0.01x_3 & = & 0, \\
 & x_3 - 0.01x_4 & = & 0, \\
 & x_1 & \geq & -10^{-9}, \\
 & x_1 & \leq & 10^{-9}, \\
 & x_4 & \geq & 10^{-4}.
\end{array}$$

Now the **MOSEK** presolve will, for the sake of efficiency, fix variables (and constraints) that have tight bounds where tightness is controlled by the parameter *dparam.presolve_tol_x*. Since the bounds

$$-10^{-9} \leq x_1 \leq 10^{-9}$$

are tight, presolve will set $x_1 = 0$. It easy to see that this implies $x_4 = 0$, which leads to the incorrect conclusion that the problem is infeasible. However a tiny change of the value 10^{-9} makes the problem feasible. In general it is recommended to avoid ill-posed problems, but if that is not possible then one solution is to reduce parameters such as *dparam.presolve_tol_x* to say 10^{-10}. This will at least make sure that presolve does not make the undesired reduction.

8.3 Debugging infeasibility

This section contains hints for debugging problems that are unexpectedly infeasible. It is always a good idea to remove the objective, i.e. only solve a feasibility problem when debugging such issues.

8.3.1 Numerical issues

Infeasible problem status may be just an artifact of numerical issues appearing when the problem is badly-scaled, barely feasible or otherwise ill-conditioned so that it is unstable under small perturbations of the data or round-off errors. This may be visible in the solution summary if the infeasibility certificate has poor quality. See Sec. 8.1.2 for how to diagnose that and Sec. 8.2 for possible hints. Sec. 8.2.3 contains examples of situations which may lead to infeasibility for numerical reasons.

We refer to Sec. 8.2 for further information on dealing with those sort of issues. For the rest of this section we concentrate on the case when the solution summary leaves little doubt that the problem solved by the optimizer actually is infeasible.

8.3.2 Locating primal infeasibility

As an example of a primal infeasible problem consider minimizing the cost of transportation between a number of production plants and stores: Each plant produces a fixed number of goods, and each store has a fixed demand that must be met. Supply, demand and cost of transportation per unit are given in Fig. 8.1.

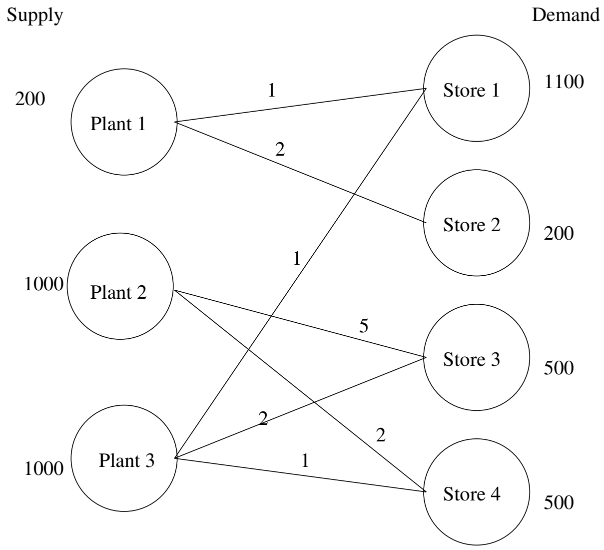


Fig. 8.1: Supply, demand and cost of transportation.

The problem represented in Fig. 8.1 is infeasible, since the total demand

$$2300 = 1100 + 200 + 500 + 500$$

exceeds the total supply

$$2200 = 200 + 1000 + 1000$$

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

$$
\begin{array}{llccccccccccccccl}
\text{minimize} & & x_{11} & + & 2x_{12} & + & 5x_{23} & + & 2x_{24} & + & x_{31} & + & 2x_{33} & + & x_{34} & & \\
\text{subject to} & s_0: & x_{11} & + & x_{12} & & & & & & & & & & & \leq & 200, \\
& s_1: & & & & & x_{23} & + & x_{24} & & & & & & & \leq & 1000, \\
& s_2: & & & & & & & & & x_{31} & + & x_{33} & + & x_{34} & \leq & 1000, \\
& d_1: & x_{11} & & & & & & & + & x_{31} & & & & & = & 1100, \\
& d_2: & & & x_{12} & & & & & & & & & & & = & 200, \\
& d_3: & & & & & x_{23} & + & & & & & x_{33} & & & = & 500, \\
& d_4: & & & & & & & x_{24} & + & & & & & x_{34} & = & 500, \\
& & & & & & & & & & & & & & x_{ij} & \geq & 0.
\end{array}
\tag{8.1}
$$

Solving problem (8.1) using **MOSEK** will result in an infeasibility status. The infeasibility certificate is contained in the dual variables an can be accessed from an API. The variables and constraints with nonzero solution values form an infeasible subproblem, which frequently is very small. See Sec. 12.1.2 or Sec. 12.2.2 for detailed specifications of infeasibility certificates.

A short infeasibility report can also be printed to the log stream. It can be turned on by setting the parameter *iparam.infeas_report_auto* to *onoffkey.on*. This causes **MOSEK** to print a report on variables and constraints which are involved in infeasibility in the above sense, i.e. have nonzero values in the certificate. The parameter *iparam.infeas_report_level* controls the amount of information presented in the infeasibility report. The default value is 1. For the above example the report is

```
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 corresponding to constraints and variables. It is a selection of those lines from the problem solution which are important in understanding primal infeasibility. In this case the constraints `s0`, `s2`, `d1`, `d2` and variables `x33`, `x34` are of importance because of nonzero dual values. The columns `Dual lower` and `Dual upper` contain the values of dual variables s_l^c, s_u^c, s_l^x and s_u^x in the primal infeasibility certificate (see Sec. 12.1.2).

In our example the certificate means that an appropriate linear combination of constraints `s0`, `s1` with coefficient $s_u^c = 1$, constraints `d1` and `d2` with coefficient $s_u^c - s_l^c = 0 - 1 = -1$ and lower bounds on `x33` and `x34` with coefficient $-s_l^x = -1$ gives a contradiction. Indeed, the combination of the four involved constraints is $x_{33} + x_{34} \leq -100$ (as indicated in the introduction, the difference between supply and demand).

It is also possible to extract the infeasible subproblem with the command-line tool. For an infeasible problem called `infeas.lp` the command:

```
mosek -d MSK_IPAR_INFEAS_REPORT_AUTO MSK_ON infeas.lp -info rinfeas.lp
```

will produce the file `rinfeas.bas.inf.lp` which contains the infeasible subproblem. Because of its size it may be easier to work with than the original problem file.

Returning to the transportation example, we discover that removing the fifth constraint $x_{12} = 200$ makes the problem feasible. Almost all undesired infeasibilities should be fixable at the modeling stage.

8.3.3 Locating dual infeasibility

A problem may also be *dual infeasible*. In this case the primal problem is usually unbounded, meaning that feasible solutions exists such that the objective tends towards infinity. For example, consider the problem

$$
\begin{array}{ll}
\text{maximize} & 200y_1 + 1000y_2 + 1000y_3 + 1100y_4 + 200y_5 + 500y_6 + 500y_7 \\
\text{subject to} & y_1 + y_4 \leq 1,\ y_1 + y_5 \leq 2,\ y_2 + y_6 \leq 5,\ y_2 + y_7 \leq 2, \\
 & y_3 + y_4 \leq 1,\ y_3 + y_6 \leq 2,\ y_3 + y_7 \leq 1 \\
 & y_1, y_2, y_3 \leq 0
\end{array}
$$

which is dual to (8.1) (and therefore is dual infeasible). The dual infeasibility report may look as follows:

```
MOSEK DUAL INFEASIBILITY REPORT.

Problem status: The problem is dual infeasible

The following constraints are involved in the infeasibility.

Index     Name              Activity         Objective        Lower bound      Upper bound
5         x33               -1.000000e+00                     NONE             2.000000e+00
6         x34               -1.000000e+00                     NONE             1.000000e+00

The following variables are involved in the infeasibility.

Index     Name              Activity         Objective        Lower bound      Upper bound
0         y1                -1.000000e+00    2.000000e+02     NONE             0.000000e+00
2         y3                -1.000000e+00    1.000000e+03     NONE             0.000000e+00
3         y4                1.000000e+00     1.100000e+03     NONE             NONE
4         y5                1.000000e+00     2.000000e+02     NONE             NONE

Interior-point solution summary
  Problem status  : DUAL_INFEASIBLE
  Solution status : DUAL_INFEASIBLE_CER
  Primal.  obj: 1.0000000000e+02    nrm: 1e+00    Viol.  con: 0e+00    var: 0e+00
```

In the report we see that the variables `y1`, `y3`, `y4`, `y5` and two constraints contribute to infeasibility with non-zero values in the `Activity` column. Therefore

$$(y_1, \ldots, y_7) = (-1, 0, -1, 1, 1, 0, 0)$$

is the dual infeasibility certificate as in Sec. 12.1.2. This just means, that along the ray

$$(0, 0, 0, 0, 0, 0, 0) + t(y_1, \ldots, y_7) = (-t, 0, -t, t, t, 0, 0),\ t > 0,$$

which belongs to the feasible set, the objective value $100t$ can be arbitrarily large, i.e. the problem is unbounded.

In the example problem we could

- Add a lower bound on `y3`. This will directly invalidate the certificate of dual infeasibility.
- Increase the objective coefficient of `y3`. Changing the coefficients sufficiently will invalidate the inequality $c^T y^* > 0$ and thus the certificate.

8.3.4 Suggestions

Primal infeasibility

When trying to understand what causes the unexpected primal infeasible status use the following hints:

- Remove the objective function. This does not change the infeasibility status but simplifies the problem, eliminating any possibility of issues related to the objective function.

- Remove cones, semidefinite variables and integer constraints. Solve only the linear part of the problem. Typical simple modeling errors will lead to infeasibility already at this stage.
- Consider whether your problem has some obvious necessary conditions for feasibility and examine if these are satisfied, e.g. total supply should be greater than or equal to total demand.
- Verify that coefficients and bounds are reasonably sized in your problem.
- See if there are any obvious contradictions, for instance a variable is bounded both in the variables and constraints section, and the bounds are contradictory.
- Consider replacing suspicious equality constraints by inequalities. For instance, instead of $x_{12} = 200$ see what happens for $x_{12} \geq 200$ or $x_{12} \leq 200$.
- Relax bounds of the suspicious constraints or variables.
- For integer problems, remove integrality constraints on some/all variables and see if the problem solves.
- Remember that variables without explicitly initialized bounds are fixed at zero.
- Form an **elastic model**: allow to violate constraints at a cost. Introduce slack variables and add them to the objective as penalty. For instance, suppose we have a constraint

$$\begin{array}{ll} \text{minimize} & c^T x, \\ \text{subject to} & a^T x \leq b. \end{array}$$

  which might be causing infeasibility. Then create a new variable y and form the problem which contains:

$$\begin{array}{ll} \text{minimize} & c^T x + y, \\ \text{subject to} & a^T x \leq b + y. \end{array}$$

  Solving this problem will reveal by how much the constraint needs to be relaxed in order to become feasible. This is equivalent to inspecting the infeasibility certificate but may be more intuitive.
- If you think you have a feasible solution or its part, fix all or some of the variables to those values. Presolve will propagate them through the model and potentially reveal more localized sources of infeasibility.
- Dump the problem in OPF or LP format and verify that the problem that was passed to the optimizer corresponds to the problem expressed in the high-level modeling language, if any such was used.

Dual infeasibility

When trying to understand what causes the unexpected dual infeasible status use the following hints:

- Verify that the objective coefficients are reasonably sized.
- Check if no bounds and constraints are missing, for example if all variables that should be nonnegative have been declared as such etc.
- Strengthen bounds of the suspicious constraints or variables.
- Remember that constraints without explicitly initialized bounds are free (no bound).
- Form an series of models with decreasing bounds on the objective, that is, instead of objective

$$\text{minimize } c^T x$$

  solve the problem with an additional constraint such as

$$c^T x = -10^5$$

  and inspect the solution to figure out the mechanism behind arbitrarily decreasing objective values. This is equivalent to inspecting the infeasibility certificate but may be more intuitive.

- Dump the problem in OPF or LP format and verify that the problem that was passed to the optimizer corresponds to the problem expressed in the high-level modeling language, if any such was used.

Please note that modifying the problem to invalidate the reported certificate does *not* imply that the problem becomes feasible — the reason for infeasibility may simply *move*, resulting a problem that is still infeasible, but for a different reason. More often, the reported certificate can be used to give a hint about errors or inconsistencies in the model that produced the problem.

8.4 Python Console

The **MOSEK** Python Console is an alternative to the **MOSEK** Command Line Tool. It can be used for interactive loading, solving and debugging optimization problems stored in files, for example **MOSEK** task files. It facilitates debugging techniques described in Sec. 8.

8.4.1 Usage

The tool requires Python 2 or 3. The **MOSEK** interface for Python must be installed following the installation instructions for Python API or Python Fusion API. In the basic case it should be sufficient to execute the script

```
python setup.py install --user
```

in the directory containing the **MOSEK** Python module.

The Python Console is contained in the file `mosekconsole.py` in the folder with **MOSEK** binaries. It can be copied to an arbitrary location. The file is also available for `download here (mosekconsole.py)`.

To run the console in interactive mode use

```
python mosekconsole.py
```

To run the console in batch mode provide a semicolon-separated list of commands as the second argument of the script, for example:

```
python mosekconsole.py "read data.task.gz; solve form=dual; writesol data"
```

The script is written using the **MOSEK** Python API and can be extended by the user if more specific functionality is required. We refer to the documentation of the Python API.

8.4.2 Examples

To read a problem from `data.task.gz`, solve it, and write solutions to `data.sol`, `data.bas` or `data.itg`:

```
read data.task.gz; solve; writesol data
```

To convert between file formats:

```
read data.task.gz; write data.mps
```

To set a parameter before solving:

```
read data.task.gz; param INTPNT_CO_TOL_DFEAS 1e-9; solve"
```

To list parameter values related to the mixed-integer optimizer in the task file:

```
read data.task.gz; param MIO
```

To print a summary of problem structure:

```
read data.task.gz; anapro
```

To solve a problem forcing the dual and switching off presolve:

```
read data.task.gz; solve form=dual presolve=no
```

To write an infeasible subproblem to a file for debugging purposes:

```
read data.task.gz; solve; infsub; write inf.opf
```

8.4.3 Full list of commands

Below is a brief description of all the available commands. Detailed information about a specific command `cmd` and its options can be obtained with

```
help cmd
```

Table 8.1: List of commands of the MOSEK Python Console.

Command	Description
`help [command]`	Print list of commands or info about a specific command
`log filename`	Save the session to a file
`intro`	Print MOSEK splashscreen
`testlic`	Test the license system
`read filename`	Load problem from file
`reread`	Reload last problem file
`solve [options]`	Solve current problem
`write filename`	Write current problem to file
`param [name [value]]`	Set a parameter or get parameter values
`paramdef`	Set all parameters to default values
`info [name]`	Get an information item
`anapro`	Analyze problem data
`hist`	Plot a histogram of problem data
`histsol`	Plot a histogram of the solutions
`spy`	Plot the sparsity pattern of the A matrix
`truncate epsilon`	Truncate small coefficients down to 0
`anasol`	Analyze solutions
`removeitg`	Remove integrality constraints
`infsub`	Replace current problem with its infeasible subproblem
`writesol basename`	Write solution(s) to file(s) with given basename
`delsol`	Remove all solutions from the task
`exit`	Leave

Chapter 9

Advanced Numerical Tutorials

9.1 Solving Linear Systems Involving the Basis Matrix

A linear optimization problem always has an optimal solution which is also a basic solution. In an optimal basic solution there are exactly m basic variables where m is the number of rows in the constraint matrix A. Define

$$B \in \mathbb{R}^{m \times m}$$

as a matrix consisting of the columns of A corresponding to the basic variables. The basis matrix B is always non-singular, i.e.

$$\det(B) \neq 0$$

or, equivalently, B^{-1} exists. This implies that the linear systems

$$B\bar{x} = w \tag{9.1}$$

and

$$B^T\bar{x} = w \tag{9.2}$$

each have a unique solution for all w.

MOSEK provides functions for solving the linear systems (9.1) and (9.2) for an arbitrary w.

In the next sections we will show how to use **MOSEK** to

- *identify the solution basis*,
- *solve arbitrary linear systems*.

9.1.1 Basis identification

To use the solutions to (9.1) and (9.2) it is important to know how the basis matrix B is constructed.

Internally **MOSEK** employs the linear optimization problem

$$\begin{array}{lccccl} \text{maximize} & & & c^T x & & \\ \text{subject to} & & & Ax - x^c & = & 0, \\ & l^x & \leq & x & \leq & u^x, \\ & l^c & \leq & x^c & \leq & u^c. \end{array} \tag{9.3}$$

where

$$x^c \in \mathbb{R}^m \text{ and } x \in \mathbb{R}^n.$$

The basis matrix is constructed of m columns taken from

$$\left[\begin{array}{cc} A & -I \end{array} \right].$$

If variable x_j is a basis variable, then the j-th column of A, denoted $a_{:,j}$, will appear in B. Similarly, if x_i^c is a basis variable, then the i-th column of $-I$ will appear in the basis. The ordering of the basis variables and therefore the ordering of the columns of B is arbitrary. The ordering of the basis variables may be retrieved by calling the function *Task.initbasissolve*. This function initializes data structures for later use and returns the indexes of the basic variables in the array `basis`. The interpretation of the `basis` is as follows. If we have

$$\texttt{basis}[i] < \texttt{numcon}$$

then the i-th basis variable is

$$x^c_{\texttt{basis}[i]}.$$

Moreover, the i-th column in B will be the i-th column of $-I$. On the other hand if

$$\texttt{basis}[i] \geq \texttt{numcon},$$

then the i-th basis variable is the variable

$$x_{\texttt{basis}[i]-\texttt{numcon}}$$

and the i-th column of B is the column

$$A_{:,(\texttt{basis}[i]-\texttt{numcon})}.$$

For instance if $\texttt{basis}[0] = 4$ and $\texttt{numcon} = 5$, then since $\texttt{basis}[0] < \texttt{numcon}$, the first basis variable is x_4^c. Therefore, the first column of B is the fourth column of $-I$. Similarly, if $\texttt{basis}[1] = 7$, then the second variable in the basis is $x_{\texttt{basis}[1]-\texttt{numcon}} = x_2$. Hence, the second column of B is identical to $a_{:,2}$.

An example

Consider the linear optimization problem:

$$\begin{array}{lccc} \text{minimize} & x_0 + x_1 & & \\ \text{subject to} & x_0 + 2x_1 & \leq & 2, \\ & x_0 + x_1 & \leq & 6, \\ & x_0, x_1 \geq 0. & & \end{array} \tag{9.4}$$

Suppose a call to *Task.initbasissolve* returns an array `basis` so that

```
basis[0] = 1,
basis[1] = 2.
```

Then the basis variables are x_1^c and x_0 and the corresponding basis matrix B is

$$\left[\begin{array}{cc} 0 & 1 \\ -1 & 1 \end{array}\right].$$

Please note the ordering of the columns in B .

Listing 9.1: A program showing how to identify the basis.

```python
import mosek

def streamprinter(text):
    sys.stdout.write(text)
    sys.stdout.flush()

def main():
    numcon = 2
    numvar = 2
```

(continues on next page)

(continued from previous page)

```
# Since the value infinity is never used, we define
# 'infinity' symbolic purposes only
infinity = 0

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

blc = [-infinity,
       -infinity]
buc = [2.0,
       6.0]

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

bux = [+infinity,
       +infinity]
w1 = [2.0, 6.0]
w2 = [1.0, 0.0]

try:
    with mosek.Env() as env:
        with env.Task(0, 0) as task:
            task.set_Stream(mosek.streamtype.log, streamprinter)
            task.inputdata(numcon, numvar,
                           c,
                           0.0,
                           ptrb,
                           ptre,
                           asub,
                           aval,
                           bkc,
                           blc,
                           buc,
                           bkx,
                           blx,
                           bux)
            task.putobjsense(mosek.objsense.maximize)
            r = task.optimize()
            if r != mosek.rescode.ok:
                print("Mosek warning:", r)

            basis = [0] * numcon
            task.initbasissolve(basis)

            #List basis variables corresponding to columns of B
            varsub = [0, 1]

            for i in range(numcon):
                if basis[varsub[i]] < numcon:
                    print("Basis variable no %d is xc%d" % (i, basis[i]))
```

(continues on next page)

(continued from previous page)

```
                    else:
                        print("Basis variable no %d is x%d" %
                              (i, basis[i] - numcon))

                # solve Bx = w1
                # varsub contains index of non-zeros in b.
                #  On return b contains the solution x and
                # varsub the index of the non-zeros in x.
                nz = 2

                nz = task.solvewithbasis(0, nz, varsub, w1)
                print("nz = %s" % nz)
                print("Solution to Bx = w1:")

                for i in range(nz):
                    if basis[varsub[i]] < numcon:
                        print("xc %s = %s" % (basis[varsub[i]], w1[varsub[i]]))
                    else:
                        print("x%s = %s" %
                              (basis[varsub[i]] - numcon, w1[varsub[i]]))

                # Solve B^Tx = w2
                nz = 1
                varsub[0] = 0

                nz = task.solvewithbasis(1, nz, varsub, w2)

                print("Solution to B^Tx = w2:")

                for i in range(nz):
                    if basis[varsub[i]] < numcon:
                        print("xc %s = %s" % (basis[varsub[i]], w2[varsub[i]]))
                    else:
                        print("x %s = %s" %
                              (basis[varsub[i]] - numcon, w2[varsub[i]]))
    except Exception as e:
        print(e)

if __name__ == '__main__':
    main()
```

In the example above the linear system is solved using the optimal basis for (9.4) and the original right-hand side of the problem. Thus the solution to the linear system is the optimal solution to the problem. When running the example program the following output is produced.

```
basis[0] = 1
Basis variable no 0 is xc1.
basis[1] = 2
Basis variable no 1 is x0.

Solution to Bx = b:

x0 = 2.000000e+00
xc1 = -4.000000e+00

Solution to B^Tx = c:

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

Please note that the ordering of the basis variables is

$$\begin{bmatrix} x_1^c \\ x_0 \end{bmatrix}$$

and thus the basis is given by:

$$B = \begin{bmatrix} 0 & 1 \\ -1 & 1 \end{bmatrix}$$

It can be verified that

$$\begin{bmatrix} x_1^c \\ x_0 \end{bmatrix} = \begin{bmatrix} -4 \\ 2 \end{bmatrix}$$

is a solution to

$$\begin{bmatrix} 0 & 1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} x_1^c \\ x_0 \end{bmatrix} = \begin{bmatrix} 2 \\ 6 \end{bmatrix}.$$

9.1.2 Solving arbitrary linear systems

MOSEK can be used to solve an arbitrary (rectangular) linear system

$$Ax = b$$

using the *Task.solvewithbasis* function without optimizing the problem as in the previous example. This is done by setting up an A matrix in the task, setting all variables to basic and calling the *Task.solvewithbasis* function with the b vector as input. The solution is returned by the function.

An example

Below we demonstrate how to solve the linear system

$$\begin{bmatrix} 0 & 1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \tag{9.5}$$

with two inputs $b = (1, -2)$ and $b = (7, 0)$.

```
import mosek

def setup(task,
          aval,
          asub,
          ptrb,
          ptre,
          numvar,
          basis):
    # Since the value infinity is never used, we define
    # 'infinity' symbolic purposes only
    infinity = 0

    skx = [mosek.stakey.bas] * numvar
    skc = [mosek.stakey.fix] * numvar

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

    for i in range(len(asub)):
        task.putacol(i, asub[i], aval[i])

    for i in range(numvar):
```

(continues on next page)

(continued from previous page)

```
        task.putconbound(i, mosek.boundkey.fx, 0.0, 0.0)

    for i in range(numvar):
        task.putvarbound(i,
                         mosek.boundkey.fr,
                         -infinity,
                         infinity)

    # Define a basic solution by specifying
    # status keys for variables & constraints.
    task.deletesolution(mosek.soltype.bas);

    task.putskcslice(mosek.soltype.bas, 0, numvar, skc);
    task.putskxslice(mosek.soltype.bas, 0, numvar, skx);

    task.initbasissolve(basis);


def main():
    numcon = 2
    numvar = 2

    aval = [[-1.0],
            [1.0, 1.0]]
    asub = [[1],
            [0, 1]]

    ptrb = [0, 1]
    ptre = [1, 3]

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

    with mosek.Env() as env:
        with mosek.Task(env) as task:
            # Directs the log task stream to the user specified
            # method task_msg_obj.streamCB
            task.set_Stream(mosek.streamtype.log,
                            lambda msg: sys.stdout.write(msg))
            # Put A matrix and factor A.
            # Call this function only once for a given task.

            basis = [0] * numvar
            b = [0.0, -2.0]
            bsub = [0, 1]

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

            # now solve rhs
            b = [1, -2]
            bsub = [0, 1]
            nz = task.solvewithbasis(0, 2, bsub, b)
            print("\nSolution to Bx = b:\n")
```

(continues on next page)

(continued from previous page)

```
            # Print solution and show correspondents
            # to original variables in the problem
            for i in range(nz):
                if basis[bsub[i]] < numcon:
                    print("This should never happen")
                else:
                    print("x%d = %d" % (basis[bsub[i]] - numcon, b[bsub[i]]))

            b[0] = 7
            bsub[0] = 0

            nz = task.solvewithbasis(0, 1, bsub, b)

            print("\nSolution to Bx = b:\n")
            # Print solution and show correspondents
            # to original variables in the problem
            for i in range(nz):
                if basis[bsub[i]] < numcon:
                    print("This should never happen")
                else:
                    print("x%d = %d" % (basis[bsub[i]] - numcon, b[bsub[i]]))

if __name__ == "__main__":
    try:
        main()
    except:
        import traceback
        traceback.print_exc()
```

The most important step in the above example is the definition of the basic solution, where we define the status key for each variable. The actual values of the variables are not important and can be selected arbitrarily, so we set them to zero. All variables corresponding to columns in the linear system we want to solve are set to basic and the slack variables for the constraints, which are all non-basic, are set to their bound.

The program produces the output:

```
Solution to Bx = b:

x1 = 1
x0 = 3

Solution to Bx = b:

x1 = 7
x0 = 7
```

9.2 Calling BLAS/LAPACK Routines from MOSEK

Sometimes users need to perform linear algebra operations that involve dense matrices and vectors. Also **MOSEK** extensively uses high-performance linear algebra routines from the BLAS and LAPACK packages and some of these routines are included in the package shipped to the users.

The **MOSEK** versions of BLAS/LAPACK routines:

- use **MOSEK** data types and return value conventions,
- preserve the BLAS/LAPACK naming convention.

Therefore the user can leverage on efficient linear algebra routines, with a simplified interface, with no need for additional packages.

List of available routines

Table 9.1: BLAS routines available.

BLAS Name	**MOSEK** function	Math Expression
AXPY	*Env.axpy*	$y = \alpha x + y$
DOT	*Env.dot*	$x^T y$
GEMV	*Env.gemv*	$y = \alpha Ax + \beta y$
GEMM	*Env.gemm*	$C = \alpha AB + \beta C$
SYRK	*Env.syrk*	$C = \alpha AA^T + \beta C$

Table 9.2: LAPACK routines available.

LAPACK Name	**MOSEK** function	Description
POTRF	*Env.potrf*	Cholesky factorization of a semidefinite symmetric matrix
SYEVD	*Env.syevd*	Eigenvalues and eigenvectors of a symmetric matrix
SYEIG	*Env.syeig*	Eigenvalues of a symmetric matrix

Source code examples

In Listing 9.2 we provide a simple working example. It has no practical meaning except showing how to organize the input and call the methods.

Listing 9.2: Calling BLAS and LAPACK routines from Optimizer API for Python.

```
import mosek

def print_matrix(x, r, c):
    for i in range(r):
        print([x[j * r + i] for j in range(c)])

with mosek.Env() as env:

    n = 3
    m = 2
    k = 3

    alpha = 2.0
    beta = 0.5

    x = [1.0, 1.0, 1.0]
    y = [1.0, 2.0, 3.0]
    z = [1.0, 1.0]
    v = [0.0, 0.0]
    #A has m=2 rows and k=3 cols
    A = [1.0, 1.0, 2.0, 2.0, 3., 3.]
    #B has k=3 rows and n=3 cols
    B = [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
    C = [0.0 for i in range(n * m)]
    D = [1.0, 1.0, 1.0, 1.0]
    Q = [1.0, 0.0, 0.0, 2.0]

# BLAS routines

    xy = env.dot(n, x, y)
    print("dot results= %f\n" % xy)

    env.axpy(n, alpha, x, y)
    print("\naxpy results is ")
    print_matrix(y, 1, len(y))
```

(continues on next page)

```
    env.gemv(mosek.transpose.no, m, n, alpha, A, x, beta, z)
    print("\ngemv results is ")
    print_matrix(z, 1, len(z))

    env.gemm(mosek.transpose.no, mosek.transpose.no,
             m, n, k, alpha, A, B, beta, C)
    print("\ngemm results is ")
    print_matrix(C, m, n)

    env.syrk(mosek.uplo.lo, mosek.transpose.no, m, k, alpha, A, beta, D)
    print("\nsyrk results is")
    print_matrix(D, m, m)

# LAPACK routines

    env.potrf(mosek.uplo.lo, m, Q)
    print("\npotrf results is ")
    print_matrix(Q, m, m)


    env.syeig(mosek.uplo.lo, m, Q, v)
    print("\nsyeig results is")
    print_matrix(v, 1, m)


    env.syevd(mosek.uplo.lo, m, Q, v)
    print("\nsyevd results is")
    print('v: ')
    print_matrix(v, 1, m)
    print('Q: ')
    print_matrix(Q, m, m)

    print("Exiting...")
```

9.3 Computing a Sparse Cholesky Factorization

Given a positive semidefinite symmetric (PSD) matrix

$$A \in \mathbb{R}^{n \times n}$$

it is well known there exists a matrix L such that

$$A = LL^T.$$

If the matrix L is lower triangular then it is called a *Cholesky factorization*. Given A is positive definite (nonsingular) then L is also nonsingular. A Cholesky factorization is useful for many reasons:

- A system of linear equations $Ax = b$ can be solved by first solving the lower triangular system $Ly = b$ followed by the upper triangular system $L^T x = y$.
- A quadratic term $x^T Ax$ in a constraint or objective can be replaced with $y^T y$ for $y = L^T x$, potentially leading to a more robust formulation (see *[And13]*).

Therefore, **MOSEK** provides a function that can compute a Cholesky factorization of a PSD matrix. In addition a function for solving linear systems with a nonsingular lower or upper triangular matrix is available.

In practice A may be very large with n is in the range of millions. However, then A is typically sparse which means that most of the elements in A are zero, and sparsity can be exploited to reduce the cost of computing the Cholesky factorization. The computational savings depend on the positions of zeros in A. For example, below a matrix A is given together with a Cholesky factor up to 5 digits of accuracy:

$$A = \begin{bmatrix} 4 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}, \quad L = \begin{bmatrix} 2.0000 & 0 & 0 & 0 \\ 0.5000 & 0.8660 & 0 & 0 \\ 0.5000 & -0.2887 & 0.8165 & 0 \\ 0.5000 & -0.2887 & -0.4082 & 0.7071 \end{bmatrix}. \tag{9.6}$$

However, if we symmetrically permute the rows and columns of A using a permutation matrix P

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}, \quad A' = PAP^T = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 4 \end{bmatrix},$$

then the Cholesky factorization of $A' = L'L'^T$ is

$$L' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

which is sparser than L.

Computing a permutation matrix that leads to the sparsest Cholesky factorization or the minimal amount of work is NP-hard. Good permutations can be chosen by using heuristics, such as the minimum degree heuristic and variants. The function *Env.computesparsecholesky* provided by **MOSEK** for computing a Cholesky factorization has a build in permutation aka. reordering heuristic. The following code illustrates the use of *Env.computesparsecholesky* and *Env.sparsetriangularsolvedense*.

Listing 9.3: How to use the sparse Cholesky factorization routine available in **MOSEK**.

```
try:
    perm, diag, lnzc, lptrc, lensubnval, lsubc, lvalc = env.computesparsecholesky(
        0,       #Disable multithread
        1,       #User reordering heuristic
        1.0e-14,#Singularity tolerance
        anzc, aptrc, asubc, avalc)

    printsparse(n, perm, diag, lnzc, lptrc, lensubnval, lsubc, lvalc)

    x = [b[p] for p in perm]    # Permuted b is stored as x.

    # Compute inv(L)*x.
    env.sparsetriangularsolvedense(mosek.transpose.no,
                                   lnzc, lptrc, lsubc, lvalc, x)

    # Compute inv(L^T)*x.
    env.sparsetriangularsolvedense(mosek.transpose.yes,
                                   lnzc, lptrc, lsubc, lvalc, x)

    print("\nSolution Ax=b: x = ", numpy.array(
        [x[j] for i in range(n) for j in range(n) if perm[j] == i]), "\n")
except:
    raise
```

We can set up the data to recreate the matrix A from (9.6):

```
# Observe that anzc, aptrc, asubc and avalc only specify the lower
# triangular part.
n = 4
anzc = [4, 1, 1, 1]
```

(continues on next page)

(continued from previous page)

```
    asubc = [0, 1, 2, 3, 1, 2, 3]
    aptrc = [0, 4, 5, 6]
    avalc = [4.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
    b = [13.0, 3.0, 4.0, 5.0]
```

and we obtain the following output:

```
Example with positive definite A.
P = [ 3 2 0 1 ]
diag(D) = [ 0.00 0.00 0.00 0.00 ]
L=
1.00 0.00 0.00 0.00
0.00 1.00 0.00 0.00
1.00 1.00 1.41 0.00
0.00 0.00 0.71 0.71

Solution A x = b, x = [ 1.00 2.00 3.00 4.00 ]
```

The output indicates that with the permutation matrix

$$P = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

there is a Cholesky factorization $PAP^T = LL^T$, where

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1.4142 & 0 \\ 0 & 0 & 0.7071 & 0.7071 \end{bmatrix}$$

The remaining part of the code solvers the linear system $Ax = b$ for $b = [13, 3, 4, 5]^T$. The solution is reported to be $x = [1, 2, 3, 4]^T$, which is correct.

The second example shows what happens when we compute a sparse Cholesky factorization of a singular matrix. In this example A is a rank 1 matrix

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}^T \tag{9.7}$$

```
    #Example 2 - singular A
    n = 3
    anzc = [3, 2, 1]
    asubc = [0, 1, 2, 1, 2, 2]
    aptrc = [0, 3, 5]
    avalc = [1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
```

Now we get the output

```
P = [ 0 2 1 ]
diag(D) = [ 0.00e+00 1.00e-14 1.00e-14 ]
L=
1.00e+00 0.00e+00 0.00e+00
1.00e+00 1.00e-07 0.00e+00
1.00e+00 0.00e+00 1.00e-07
```

which indicates the decomposition

$$PAP^T = LL^T - D$$

where

$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \quad L = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 10^{-7} & 0 \\ 1 & 0 & 10^{-7} \end{bmatrix}, \quad D = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 10^{-14} & 0 \\ 0 & 0 & 10^{-14} \end{bmatrix}.$$

Since A is only positive semdefinite, but not of full rank, some of diagonal elements of A are boosted to make it truely positive definite. The amount of boosting is passed as an argument to *Env.computesparsecholesky*, in this case 10^{-14}. Note that

$$PAP^T = LL^T - D$$

where D is a small matrix so the computed Cholesky factorization is exact of slightly perturbed A. In general this is the best we can hope for in finite precision and when A is singular or close to being singular.

We will end this section by a word of caution. Computing a Cholesky factorization of a matrix that is not of full rank and that is not suffciently well conditioned may lead to incorrect results i.e. a matrix that is indefinite may declared positive semidefinite and vice versa.

9.4 Converting a quadratically constrained problem to conic form

MOSEK employs the following form of quadratic problems:

$$\begin{array}{lccccl} \text{minimize} & & \frac{1}{2}x^TQ^ox + c^Tx + c^f & & & \\ \text{subject to} & l_k^c & \leq & \frac{1}{2}x^TQ^kx + \sum_{j=0}^{n-1} a_{k,j}x_j & \leq & u_k^c, \quad k = 0, \ldots, m-1, \\ & l_j^x & \leq & x_j & \leq & u_j^x, \quad j = 0, \ldots, n-1. \end{array} \tag{9.8}$$

A conic quadratic constraint has the form

$$x \in \mathcal{Q}^n$$

in its most basic form where

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

A quadratic problem such as (9.8), if convex, can be reformulated in conic form. This is in fact the reformulation **MOSEK** performs internally. It has many advantages:

- elegant duality theory for conic problems,
- reporting accurate dual information for quadratic inequalities is hard and/or computational expensive,
- it certifies that the original quadratic problem is indeed convex,
- modeling directly in conic form usually leads to a better model *[And13]* i.e. a faster solution time and better numerical properties.

In addition, there are more types of conic constraints that can be combined with a quadratic cone, for example semidefinite cones.

MOSEK offers a function that performs the conversion from quadratic to conic quadratic form explicitly. Note that the reformulation is not unique. The approach followed by **MOSEK** is to introduce additional variables, linear constraints and quadratic cones to obtain a larger but equivalent problem in which the original variables are preserved.

In particular:

- all variables and constraints are kept in the problem,
- each quadratic constraint and quadratic terms in the objective generate one rotated quadratic cone,

- each quadratic constraint will contain no coefficients and upper/lower bounds will be set to $\infty, -\infty$ respectively.

This allows the user to recover the original variable and constraint values, as well as their dual values, with no conversion or additional effort.

Note: *Task.toconic* modifies the input task in-place: this means that if the reformulation is not possible, i.e. the problem is not conic representable, the state of the task is in general undefined. The user should consider cloning the original task.

9.4.1 Quadratic Constraint Reformulation

Let us assume we want to convert the following quadratic constraint

$$l \leq \frac{1}{2}x^T Qx + \sum_{j=0}^{n-1} a_j x_j \leq u$$

to conic form. We first check whether $l = -\infty$ or $u = \infty$, otherwise either the constraint can be dropped, or the constraint is not convex. Thus let us consider the case

$$\frac{1}{2}x^T Qx + \sum_{j=0}^{n-1} a_j^T x_j \leq u. \tag{9.9}$$

Introducing an additional variable w such that

$$w = u - \sum_{j=0}^{n-1} a_j^T x_j \tag{9.10}$$

we obtain the equivalent form

$$\begin{array}{lcl} \frac{1}{2}x^T Qx & \leq & w, \\ u - \sum_{j=0}^{n-1} a_j x_j & = & w. \end{array}$$

If Q is positive semidefinite, then there exists a matrix F such that

$$Q = FF^T \tag{9.11}$$

and therefore we can write

$$\begin{array}{lcl} \|Fx\|^2 & \leq & 2w, \\ u - \sum_{j=0}^{n-1} a_j^T x_j & = & w. \end{array}$$

Introducing an additional variable $z = 1$, and setting $y = Fx$ we obtain the conic formulation

$$\begin{array}{ll} (w, z, y) & \in \mathcal{Q}_r, \\ z = 1 & \\ y = Fx & \\ w = u - a^T x. & \end{array} \tag{9.12}$$

Summarizing, for each quadratic constraint involving t variables, **MOSEK** introduces

1. a rotated quadratic cone of dimension $t + 2$,
2. two additional variables for the cone roots,
3. t additional variables to map the remaining part of the cone,
4. t linear constraints.

A quadratic term in the objective is reformulated in a similar fashion. We refer to *[And13]* for a more thorough discussion.

Example

Next we consider a simple problem with quadratic objective function:

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}(13x_0^2 + 17x_1^2 + 12x_2^2 + 24x_0x_1 + 12x_1x_2 - 4x_0x_2) - 22x_0 - 14.5x_1 + 12x_2 + 1 \\ \text{subject to} & -1 \leq x_0, x_1, x_2 \leq 1 \end{array}$$

We can specify it in the human-readable OPF format.

```
[comment]
An example of small QO problem from Boyd and Vandenberghe, "Convex Optimization", page 189 ex␣
↪4.3
The solution is (1,0.5,-1)
[/comment]

[variables]
x0 x1 x2
[/variables]

[objective min]
0.5 (13 x0^2  + 17 x1^2 + 12 x2^2 + 24 x0 * x1 + 12 x1 * x2 - 4 x0 * x2 ) - 22 x0 - 14.5 x1 +␣
↪12 x2 + 1
[/objective]

[bounds]
[b] -1 <= * <= 1 [/b]
[/bounds]
```

The objective function is convex, the minimum is attained for $x^\star = (1, 0.5, -1)$. The conversion will introduce first a variable x_3 in the objective function such that $x_3 \geq 1/2x^TQx$ and then convert the latter directly in conic form. The converted problem follows:

$$\begin{array}{lr} \text{minimize} & -22x_0 - 14.5x_1 + 12x_2 + x_3 + 1 \\ \text{subject to} & 3.61x_0 + 3.33x_1 - 0.55x_2 - x_6 = 0 \\ & +2.29x_1 + 3.42x_2 - x_7 = 0 \\ & 0.81x_1 - x_8 = 0 \\ & -x_3 + x_4 = 0 \\ & x_5 = 1 \\ & (x_4, x_5, x_6, x_7, x_8) \in \mathcal{Q}_\nabla \\ & -1 \leq x_0, x_1, x_2 \leq 1 \end{array}$$

The model generated by *Task.toconic* is

```
[comment]
   Written by MOSEK version 8.1.0.19
   Date 21-08-17
   Time 10:53:36
[/comment]

[hints]
  [hint NUMVAR] 9 [/hint]
  [hint NUMCON] 4 [/hint]
  [hint NUMANZ] 11 [/hint]
  [hint NUMQNZ] 0 [/hint]
  [hint NUMCONE] 1 [/hint]
[/hints]

[variables disallow_new_variables]
  x0000_x0 x0001_x1 x0002_x2 x0003 x0004
  x0005 x0006 x0007 x0008
[/variables]

[objective minimize]
```

(continues on next page)

(continued from previous page)

```
   - 2.2e+01 x0000_x0 - 1.45e+01 x0001_x1 + 1.2e+01 x0002_x2 + x0003
   + 1e+00
[/objective]

[constraints]
  [con c0000]  3.605551275463989e+00 x0000_x0 - 5.547001962252291e-01 x0002_x2 + 3.
↪328201177351375e+00 x0001_x1 - x0006 = 0e+00 [/con]
  [con c0001]  3.419401657060442e+00 x0002_x2 + 2.294598480395823e+00 x0001_x1 - x0007 = 0e+00␣
↪[/con]
  [con c0002]  8.111071056538127e-01 x0001_x1 - x0008 = 0e+00 [/con]
  [con c0003]  - x0003 + x0004 = 0e+00 [/con]
[/constraints]

[bounds]
  [b] -1e+00      <= x0000_x0,x0001_x1,x0002_x2 <= 1e+00 [/b]
  [b]                x0003,x0004 free [/b]
  [b]                x0005 =  1e+00 [/b]
  [b]                x0006,x0007,x0008 free [/b]
  [cone rquad k0000] x0004, x0005, x0006, x0007, x0008 [/cone]
[/bounds]
```

We can clearly see that constraints c0000, c0001 and c0002 represent the original linear constraints as in (9.11), while c0003 corresponds to (9.10). The cone roots are x0005 and x0004.

Chapter 10

Technical guidelines

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

10.1 Memory management and garbage collection

Users who experience memory leaks, especially:

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

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

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

```
with mosek.Env() as env:
  with env.Task(0, 0) as task:
    # Build an optimization problem
    # ...
```

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

10.2 Names

All elements of an optimization problem in **MOSEK** (objective, constraints, variables, etc.) can be given names. Assigning meaningful names to variables and constraints makes it much easier to understand and debug optimization problems dumped to a file. On the other hand, note that assigning names can substantially increase setup time, so it should be avoided in time-critical applications.

Names of various elements of the problem can be set and retrieved using various functions listed in the **Names** section of Sec. 15.2.

10.3 Multithreading

Thread safety

Sharing a task between threads is safe, as long as it is not accessed from more than one thread at a time. Multiple tasks can be created and used in parallel without any problems.

Parallelization

The interior-point and mixed-integer optimizers in **MOSEK** are parallelized. By default **MOSEK** will automatically select the number of threads. However, the maximum number of threads allowed can be changed by setting the parameter *iparam.num_threads* and related parameters. This should never exceed the number of cores. See Sec. 13 and Sec. 13.4 for more details.

The speed-up obtained when using multiple threads is highly problem and hardware dependent. We recommend experimenting with various thread numbers to determine the optimal settings. For small problems using multiple threads may be counter-productive because of the associated overhead.

Determinism

By default the optimizer is run-to-run deterministic, which means that it will return the same answer each time it is run on the same machine with the same input, the same parameter settings (including number of threads) and no time limits.

Setting the number of threads

The number of threads the optimizer uses can be changed with the parameter *iparam.num_threads*.

For conic problems (when the conic optimizer is used) the value set at the first optimization will remain fixed through the lifetime of the process. The thread pool will be reserved once for all and subsequent changes to *iparam.num_threads* will have no effect. The only possibility at that point is to switch between multi-threaded and single-threaded interior-point optimization using the parameter *iparam.intpnt_multi_thread*.

The parameter *iparam.num_threads* affects only the optimizer. It may be the case that `numpy` is consuming more threads. In most cases this can be limited by setting the environment variable `MKL_NUM_THREADS`. See the `numpy` documentation for more details.

10.4 Efficiency

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

Reduce the number of function calls and avoid input loops

For example, instead of setting the entries in the linear constraint matrix one by one (*Task.putaij*) define them all at once (*Task.putaijlist*) or in convenient large chunks (*Task.putacollist* etc.)

Use one environment only

If possible share the environment between several tasks. For most applications you need to create only a single environment.

Read part of the solution

When fetching the solution, data has to be copied from the optimizer to the user's data structures. Instead of fetching the whole solution, consider fetching only the interesting part (see for example *Task.getxxslice* and similar).

Avoiding memory fragmentation

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

- *Task.putmaxnumvar*. Estimate for the number of variables.
- *Task.putmaxnumcon*. Estimate for the number of constraints.
- *Task.putmaxnumcone*. Estimate for the number of cones.
- *Task.putmaxnumbarvar*. Estimate for the number of semidefinite matrix variables.
- *Task.putmaxnumanz*. Estimate for the number of non-zeros in A.
- *Task.putmaxnumqnz*. Estimate for the number of non-zeros in the quadratic terms.

None of these functions changes the problem, they only serve as hints. If the problem ends up growing larger, the estimates are automatically increased.

Do not mix `put-` and `get-` functions

MOSEK will queue `put-` requests internally until a `get-` function is called. If `put-` and `get-` calls are interleaved, the queue will have to be flushed more frequently, decreasing efficiency.

In general `get-` commands should not be called often (or at all) during problem setup.

Use the LIFO principle

When removing constraints and variables, try to use a LIFO (Last In First Out) approach. **MOSEK** can more efficiently remove constraints and variables with a high index than a small index.

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

Add more constraints and variables than you need (now)

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

Do not remove basic variables

When performing re-optimizations, instead of removing a basic variable it may be more efficient to fix the variable at zero and then remove it when the problem is re-optimized and it has left the basis. This makes it easier for **MOSEK** to restart the simplex optimizer.

10.5 The license system

MOSEK is a commercial product that **always** needs a valid license to work. **MOSEK** uses a third party license manager to implement license checking. The number of license tokens provided determines the number of optimizations that can be run simultaneously.

By default a license token remains checked out from the first optimization until the end of the **MOSEK** session, i.e.

- a license token is checked out when *Task.optimize* is first called, and
- it is returned when the **MOSEK** environment is deleted.

Calling *Task.optimize* from different threads using the same **MOSEK** environment only consumes one license token.

Starting the optimization when no license tokens are available will result in an error.

Default behaviour of the license system can be changed in several ways:

- Setting the parameter *iparam.cache_license* to *onoffkey.off* will force **MOSEK** to return the license token immediately after the optimization completed.

- Setting the license wait flag with the parameter *iparam.license_wait* will force **MOSEK** to wait until a license token becomes available instead of returning with an error. The wait time between checks can be set with *Env.putlicensewait*.
- Additional license checkouts and checkins can be performed with the functions *Env.checkinlicense* and *Env.checkoutlicense*.
- Usually the license system is stopped automatically when the **MOSEK** library is unloaded. However, when the user explicitly unloads the library (using e.g. FreeLibrary), the license system must be stopped before the library is unloaded. This can be done by calling the function *Env.licensecleanup* as the last function call to **MOSEK**.

10.6 Deployment

When redistributing a Python application using the **MOSEK** Optimizer API for Python 9.1.8, the following libraries must be included:

64-bit Linux	64-bit Windows	32-bit Windows	64-bit Mac OS
`libmosek64.so.9.1`	`mosek64_9_1.dll`	`mosek9_1.dll`	`libmosek64.9.1.dylib`
`libcilkrts.so.5`	`cilkrts20.dll`	`cilkrts20.dll`	`libcilkrts.5.dylib`
`libmosekxx9_1.so`	`mosekxx9_1.dll`	`mosekxx9_1.dll`	`libmosekxx9_1.dylib`

Furthermore, one (or both) of the directories

- `python/2/mosek` for Python 2.x applications,
- `python/3/mosek` for Python 3.x applications.

must be included.

By default the **MOSEK** Python API will look for the binary libraries in the **MOSEK** module directory, i.e. the directory containing `__init__.py`. Alternatively, if the binary libraries reside in another directory, the application can pre-load the `mosekxx` library from another location before `mosek` is imported, e.g. like this

```
import ctypes ; ctypes.CDLL('my/path/to/mosekxx.dll')
```

Chapter 11

Case Studies

In this section we present some case studies in which the Optimizer API for Python is used to solve real-life applications. These examples involve some more advanced modeling skills and possibly some input data. The user is strongly recommended to first read the basic tutorials of Sec. 6 before going through these advanced case studies.

- *Portfolio Optimization*
  - **Keywords:** Markowitz model, variance, risk, efficient frontier, transaction cost, market impact cost
  - **Type:** Conic Quadratic, Power Cone, Mixed-Integer Optimization
- *Logistic regression*
  - **Keywords:** machine learning, logistic regression, classifier, log-sum-exp, softplus, regularization
  - **Type:** Exponential Cone, Quadratic Cone
- *Concurrent Optimizer*
  - **Keywords:** Concurrent optimization
  - **Type:** Linear Optimization, Mixed-Integer Optimization

11.1 Portfolio Optimization

In this section the Markowitz portfolio optimization problem and variants are implemented using the **MOSEK** optimizer API.

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

11.1.1 A Basic Portfolio Optimization Model

The classical Markowitz portfolio optimization problem considers investing in n stocks or assets held over a period of time. Let x_j denote the amount invested in asset j, and assume a stochastic model where the return of the assets is a random variable r with known mean

$$\mu = \mathbf{E} r$$

and covariance

$$\Sigma = \mathbf{E}(r - \mu)(r - \mu)^T.$$

The return of the investment is also a random variable $y = r^T x$ with mean (or expected return)

$$\mathbf{E} y = \mu^T x$$

and variance

$$\mathbf{E}(y - \mathbf{E} y)^2 = x^T \Sigma x.$$

The standard deviation

$$\sqrt{x^T \Sigma x}$$

is usually associated with risk.

The problem facing the investor is to rebalance the portfolio to achieve a good compromise between risk and expected return, e.g., maximize the expected return subject to a budget constraint and an upper bound (denoted γ) on the tolerable risk. This leads to the optimization problem

$$\begin{array}{lrcl} \mbox{maximize} & \mu^T x & & \\ \mbox{subject to} & e^T x & = & w + e^T x^0, \\ & x^T \Sigma x & \leq & \gamma^2, \\ & x & \geq & 0. \end{array} \tag{11.1}$$

The variables x denote the investment i.e. x_j is the amount invested in asset j and x_j^0 is the initial holding of asset j. Finally, w is the initial amount of cash available.

A popular choice is $x^0 = 0$ and $w = 1$ because then x_j may be interpreted as the relative amount of the total portfolio that is invested in asset j.

Since e is the vector of all ones then

$$e^T x = \sum_{j=1}^{n} x_j$$

is the total investment. Clearly, the total amount invested must be equal to the initial wealth, which is

$$w + e^T x^0.$$

This leads to the first constraint

$$e^T x = w + e^T x^0.$$

The second constraint

$$x^T \Sigma x \leq \gamma^2$$

ensures that the variance, is bounded by the parameter γ^2. Therefore, γ specifies an upper bound of the standard deviation (risk) the investor is willing to undertake. Finally, the constraint

$$x_j \geq 0$$

excludes the possibility of short-selling. This constraint can of course be excluded if short-selling is allowed.

The covariance matrix Σ is positive semidefinite by definition and therefore there exist a matrix G such that

$$\Sigma = GG^T. \tag{11.2}$$

In general the choice of G is **not** unique and one possible choice of G is the Cholesky factorization of Σ. However, in many cases another choice is better for efficiency reasons as discussed in Sec. 11.1.3. For a given G we have that

$$\begin{array}{rcl} x^T \Sigma x & = & x^T GG^T x \\ & = & \left\|G^T x\right\|^2. \end{array}$$

Hence, we may write the risk constraint as

$$\gamma \geq \left\|G^T x\right\|$$

or equivalently

$$(\gamma, G^T x) \in \mathcal{Q}^{n+1},$$

where $\mathcal{Q}^{n+1}$ is the $(n+1)$-dimensional quadratic cone. Therefore, problem (11.1) can be written as

$$\begin{array}{lrcl} \text{maximize} & \mu^T x & & \\ \text{subject to} & e^T x & = & w + e^T x^0, \\ & (\gamma, G^T x) & \in & \mathcal{Q}^{n+1}, \\ & x & \geq & 0, \end{array} \tag{11.3}$$

which is a conic quadratic optimization problem that can easily be formulated and solved with Optimizer API for Python. Subsequently we will use the example data

$$\mu = \begin{bmatrix} 0.1073 \\ 0.0737 \\ 0.0627 \end{bmatrix}$$

and

$$\Sigma = 0.1 \cdot \begin{bmatrix} 0.2778 & 0.0387 & 0.0021 \\ 0.0387 & 0.1112 & -0.0020 \\ 0.0021 & -0.0020 & 0.0115 \end{bmatrix}.$$

This implies

$$G^T = \sqrt{0.1} \begin{bmatrix} 0.5271 & 0.0734 & 0.0040 \\ 0 & 0.3253 & -0.0070 \\ 0 & 0 & 0.1069 \end{bmatrix}$$

Why a Conic Formulation?

Problem (11.1) is a convex quadratically constrained optimization problem that can be solved directly using **MOSEK**. Why then reformulate it as a conic quadratic optimization problem (11.3)? The main reason for choosing a conic model is that it is more robust and usually solves faster and more reliably. For instance it is not always easy to numerically validate that the matrix Σ in (11.1) is positive semidefinite due to the presence of rounding errors. It is also very easy to make a mistake so Σ becomes indefinite. These problems are completely eliminated in the conic formulation.

Moreover, observe the constraint

$$\left\|G^T x\right\| \leq \gamma$$

more numerically robust than

$$x^T \Sigma x \leq \gamma^2$$

for very small and very large values of γ. Indeed, if say $\gamma \approx 10^4$ then $\gamma^2 \approx 10^8$, which introduces a scaling issue in the model. Hence, using conic formulation we work with the standard deviation instead of variance, which usually gives rise to a better scaled model.

Implementing the Portfolio Model

Creating a matrix formulation

The Optimizer API for Python requires that an optimization problem is entered in the following standard form:

$$\begin{array}{lccccc} \text{maximize} & & & c^T \hat{x} & & \\ \text{subject to} & l^c & \leq & A\hat{x} & \leq & u^c, \\ & l^x & \leq & \hat{x} & \leq & u^x, \\ & & & \hat{x} \in \mathcal{K}. & & \end{array} \tag{11.4}$$

We refer to $\hat{x}$ as the API variable. It means we need to reformulate (11.3). The first step is to introduce auxiliary variables so that the conic constraint involves only unique variables:

$$\begin{array}{lrcl} \text{maximize} & \mu^T x & & \\ \text{subject to} & e^T x & = & w + e^T x^0, \\ & G^T x - t & = & 0, \\ & [s; t] & \in & \mathcal{Q}^{n+1}, \\ & x & \geq & 0, \\ & s & = & \gamma. \end{array} \tag{11.5}$$

Here s is an additional scalar variable and t is a vector variable of dimension n. The next step is to concatenate all the variables into one long variable vector:

$$\hat{x} = [x; s; t] = \left[\begin{array}{c} x \\ s \\ t \end{array}\right] \tag{11.6}$$

The details of the concatenation are specified below.

Table 11.1: Storage layout of the $\hat{x}$ variable.

Variable	Length	Offset
x	n	0
s	1	n
t	n	$n+1$

The offset determines where the variable starts. (Note that all variables are indexed from 0). For instance

$$\hat{x}_{n+1+i} = t_i.$$

because the offset of the t variable is $n+1$.

Given the ordering of the variables specified by (11.6) it is useful to visualize the linear constraints (11.4) in an explicit block matrix form:

$$
\left[\begin{array}{c|c|ccc}
1 & 0 & & 0 & \\
\hline
 & & -1 & & \\
G^T & 0 & & -1 & \\
 & & & & -1
\end{array}\right]
\cdot
\left[\begin{array}{c}
x \\
\hline
s \\
\hline
t
\end{array}\right]
=
\left[\begin{array}{c}
w + e^T x_0 \\
\hline
0
\end{array}\right]. \tag{11.7}
$$

In other words, we should define the specific components of the problem description as follows:

$$
\begin{array}{rcl}
c & = & \left[\begin{array}{ccc} \mu^T & 0 & 0_n \end{array}\right]^T, \\
A & = & \left[\begin{array}{ccc} e^T & 0 & 0_n \\ G^T & 0_n & -I_n \end{array}\right], \\
l^c & = & \left[\begin{array}{cc} w + e^T x^0 & 0_n \end{array}\right]^T, \\
u^c & = & \left[\begin{array}{cc} w + e^T x^0 & 0_n \end{array}\right]^T, \\
l^x & = & \left[\begin{array}{ccc} 0_n & \gamma & -\infty_n \end{array}\right]^T, \\
u^x & = & \left[\begin{array}{ccc} \infty_n & \gamma & \infty_n \end{array}\right]^T.
\end{array} \tag{11.8}
$$

Source code example

From the block matrix form (11.7) and the explicit specification (11.8), using the offset information in Table 11.1 it is easy to calculate the index and value of each entry of the linear constraint matrix. The code below sets up the general optimization problem (11.5) and solves it for the example data. Of course it is only necessary to set non-zero entries of the linear constraint matrix.

Listing 11.1: Code implementing model (11.5).

```
import mosek
import sys

def streamprinter(text):
    sys.stdout.write("%s" % text),

if __name__ == '__main__':

    n = 3
    gamma = 0.05
    mu = [0.1073, 0.0737, 0.0627]
    GT = [[0.1667, 0.0232, 0.0013],
          [0.0000, 0.1033, -0.0022],
          [0.0000, 0.0000, 0.0338]]
    x0 = [0.0, 0.0, 0.0]
    w = 1.0

    inf = 0.0 # This value has no significance

    with mosek.Env() as env:
        with env.Task(0, 0) as task:
            task.set_Stream(mosek.streamtype.log, streamprinter)

            # Constraints.
            task.appendcons(1 + n)

            # Total budget constraint - set bounds l^c = u^c
            rtemp = w + sum(x0)
            task.putconbound(0, mosek.boundkey.fx, rtemp, rtemp)
            task.putconname(0, "budget")
```

(continues on next page)

(continued from previous page)

```python
        # The remaining constraints GT * x - t = 0 - set bounds l^c = u^c
        task.putconboundlist(range(1 + 0, 1 + n), [mosek.boundkey.fx] * n, [0.0] * n, [0.0] * n)
        for j in range(1, 1 + n):
            task.putconname(j, "GT[%d]" % j)

        # Variables.
        task.appendvars(1 + 2 * n)

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

        # x variables.
        # Returns of assets in the objective
        task.putclist(range(offsetx + 0, offsetx + n), mu)
        # Coefficients in the first row of A
        task.putaijlist([0] * n, range(offsetx + 0, offsetx + n), [1.0] * n)
        # No short-selling - x^l = 0, x^u = inf
        task.putvarboundslice(offsetx, offsetx + n, [mosek.boundkey.lo] * n, [0.0] * n, [inf] * n)
        for j in range(0, n):
            task.putvarname(offsetx + j, "x[%d]" % (1 + j))

        # s variable is a constant equal to gamma
        task.putvarbound(offsets + 0, mosek.boundkey.fx, gamma, gamma)
        task.putvarname(offsets + 0, "s")

        # t variables (t = GT*x).
        # Copying the GT matrix in the appropriate block of A
        for j in range(0, n):
            task.putaijlist(
                [1 + j] * n, range(offsetx + 0, offsetx + n), GT[j])
        # Diagonal -1 entries in a block of A
        task.putaijlist(range(1, n + 1), range(offsett + 0, offsett + n), [-1.0] * n)
        # Free - no bounds
        task.putvarboundslice(offsett + 0, offsett + n, [mosek.boundkey.fr] * n, [-inf] * n, [inf] * n)
        for j in range(0, n):
            task.putvarname(offsett + j, "t[%d]" % (1 + j))

        # Define the cone spanned by variables (s, t), i.e. dimension = n + 1
        task.appendcone(mosek.conetype.quad, 0.0, [offsets] + list(range(offsett, offsett + n)))
        task.putconename(0, "stddev")

        task.putobjsense(mosek.objsense.maximize)

        # Dump the problem to a human readable OPF file.
        task.writedata("dump.opf")

        task.optimize()

        # Display solution summary for quick inspection of results.
        task.solutionsummary(mosek.streamtype.msg)

        # Retrieve results
        xx = [0.] * (n + 1)
        task.getxxslice(mosek.soltype.itr, offsetx + 0, offsets + 1, xx)
```

(continues on next page)

(continued from previous page)

```
            expret = sum(mu[j] * xx[j] for j in range(offsetx, offsetx + n))
            stddev = xx[offsets]

            print("\nExpected return %e for gamma %e\n" % (expret, stddev))
```

The above code produces the result:

Listing 11.2: Output from the solver.

```
Interior-point solution summary
  Problem status  : PRIMAL_AND_DUAL_FEASIBLE
  Solution status : OPTIMAL
  Primal.  obj: 7.4766507287e-02    nrm: 1e+00    Viol.  con: 2e-08    var: 0e+00    cones: 2e-
↪08
  Dual.    obj: 7.4766554102e-02    nrm: 3e-01    Viol.  con: 0e+00    var: 3e-08    cones:␣
↪0e+00

Expected return 7.476651e-02 for gamma 5.000000e-02
```

Source code comments

The source code is a direct translation of the model (11.5) using the explicit block matrix specification (11.8) but a few comments are nevertheless in place.

In the lines

```
            # Offset of variables into the API variable.
            offsetx = 0
            offsets = n
            offsett = n + 1
```

offsets into the **MOSEK** API variable are stored as in Table 11.1. The code

```
            # Returns of assets in the objective
            task.putclist(range(offsetx + 0, offsetx + n), mu)
            # Coefficients in the first row of A
            task.putaijlist([0] * n, range(offsetx + 0, offsetx + n), [1.0] * n)
            # No short-selling - x^l = 0, x^u = inf
            task.putvarboundslice(offsetx, offsetx + n, [mosek.boundkey.lo] * n, [0.0] * n,␣
↪[inf] * n)
            for j in range(0, n):
                task.putvarname(offsetx + j, "x[%d]" % (1 + j))
```

sets up the data for x variables. For instance

```
            # Returns of assets in the objective
            task.putclist(range(offsetx + 0, offsetx + n), mu)
```

inputs the objective coefficients for the x variables. Moreover, the code

```
            for j in range(0, n):
                task.putvarname(offsetx + j, "x[%d]" % (1 + j))
```

assigns meaningful names to the API variables. This is not needed but it makes debugging easier.

Note that the solution values are only accessed for the interesting variables; for instance the auxiliary variable t is omitted from this process.

Debugging Tips

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

```
        # Dump the problem to a human readable OPF file.
        task.writedata("dump.opf")
```

does that and it produces a file with the content:

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

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

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

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

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

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

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

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

11.1.2 The efficient Frontier

The portfolio computed by the Markowitz model is efficient in the sense that there is no other portfolio giving a strictly higher return for the same amount of risk. An efficient portfolio is also sometimes called a Pareto optimal portfolio. Clearly, an investor should only invest in efficient portfolios and therefore it may be relevant to present the investor with all efficient portfolios so the investor can choose the portfolio that has the desired tradeoff between return and risk.

Given a nonnegative α the problem

$$\begin{array}{ll} \text{maximize} & \mu^T x - \alpha x^T \Sigma x \\ \text{subject to} & e^T x = w + e^T x^0, \\ & x \geq 0. \end{array} \tag{11.9}$$

is one standard way to trade the expected return against penalizing variance. Note that, in contrast to the previous example, we explicitly use the variance ($\|G^T x\|_2^2$) rather than standard deviation ($\|G^T x\|_2$), therefore the conic model includes a rotated quadratic cone:

$$
\begin{array}{lrcll}
\text{maximize} & \mu^T x - \alpha s & & & \\
\text{subject to} & e^T x & = & w + e^T x^0, & \\
 & G^T x - t & = & 0, & \\
 & u & = & 0.5, & \\
 & (s, u, t) & \in & \mathcal{Q}_r^{n+2} & \quad \text{(evaluates to } s \geq \|G^T x\|_2^2 = x^T \Sigma x\text{)}, \\
 & x & \geq & 0. &
\end{array}
\tag{11.10}
$$

Ideally the problem (11.9) should be solved for all values $\alpha \geq 0$ but in practice it is impossible. Using the example data as before, the optimal values of return and variance for several values of α are shown below:

Listing 11.4: Results obtained solving problem (11.9) for different values of α.

```
alpha         exp ret       variance
0.000e+00     1.073e-01     2.779e-02
2.500e-01     1.073e-01     2.779e-02
5.000e-01     1.073e-01     2.779e-02
7.500e-01     1.057e-01     2.554e-02
1.000e+00     9.965e-02     1.851e-02
1.500e+00     8.802e-02     8.850e-03
2.000e+00     8.213e-02     5.415e-03
2.500e+00     7.860e-02     3.826e-03
3.000e+00     7.625e-02     2.963e-03
3.500e+00     7.457e-02     2.442e-03
4.000e+00     7.331e-02     2.104e-03
4.500e+00     7.232e-02     1.873e-03
```

Source code example

The example code in Listing 11.5 demonstrates how to compute the efficient portfolios for several values of α. The code is mostly similar to the one in Sec. 11.1.1, except the problem is re-optimized in a loop for varying α.

Listing 11.5: Code implementing model (11.9).

```python
import mosek

def streamprinter(text):
    print("%s" % text),

if __name__ == '__main__':

    n = 3
    mu = [0.1073, 0.0737, 0.0627]
    GT = [[0.1667, 0.0232, 0.0013],
          [0.0000, 0.1033, -0.0022],
          [0.0000, 0.0000, 0.0338]]
    x0 = [0.0, 0.0, 0.0]
    w = 1.0
    alphas = [0.0, 0.25, 0.5, 0.75, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5]

    inf = 0.0 # This value has no significance

    with mosek.Env() as env:
        with env.Task(0, 0) as task:
            task.set_Stream(mosek.streamtype.log, streamprinter)

            rtemp = w + sum(x0)
```

(continues on next page)

(continued from previous page)

```
        # Constraints.
        task.appendcons(1 + n)
        task.putconbound(0, mosek.boundkey.fx, rtemp, rtemp)
        task.putconname(0, "budget")

        task.putconboundlist(range(1 + 0, 1 + n), n *
                             [mosek.boundkey.fx], n * [0.0], n * [0.0])
        for j in range(1, 1 + n):
            task.putconname(j, "GT[%d]" % j)

        # Variables.
        task.appendvars(2 + 2 * n)

        offsetx = 0   # Offset of variable x into the API variable.
        offsets = n   # Offset of variable s into the API variable.
        offsett = n + 1 # Offset of variable t into the API variable.
        offsetu = 2*n + 1 # Offset of variable u into the API variable.

        # x variables.
        task.putclist(range(offsetx + 0, offsetx + n), mu)
        task.putaijlist(
            n * [0], range(offsetx + 0, offsetx + n), n * [1.0])
        for j in range(0, n):
            task.putaijlist(
                n * [1 + j], range(offsetx + 0, offsetx + n), GT[j])

        task.putvarboundsliceconst(offsetx, offsetx + n, mosek.boundkey.lo, 0.0, inf)

        for j in range(0, n):
            task.putvarname(offsetx + j, "x[%d]" % (1 + j))

        # s variable.
        task.putvarbound(offsets + 0, mosek.boundkey.fr, -inf, inf)
        task.putvarname(offsets + 0, "s")

        # u variable.
        task.putvarbound(offsetu + 0, mosek.boundkey.fx, 0.5, 0.5)
        task.putvarname(offsetu + 0, "u")

        # t variables.
        task.putaijlist(range(1, n + 1), range(offsett +
                                               0, offsett + n), n * [-1.0])
        task.putvarboundsliceconst(offsett, offsett + n, mosek.boundkey.fr, -inf, inf)
        for j in range(0, n):
            task.putvarname(offsett + j, "t[%d]" % (1 + j))

        task.appendcone(mosek.conetype.rquad, 0.0,
                        [offsets, offsetu] + list(range(offsett, offsett + n)))
        task.putconename(0, "variance")

        task.putobjsense(mosek.objsense.maximize)

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

        for alpha in alphas:
            # Dump the problem to a human readable OPF file.
            #task.writedata("dump.opf")

            task.putcj(offsets + 0, -alpha)
```

(continues on next page)

```
                task.optimize()

                # Display the solution summary for quick inspection of results.
                # task.solutionsummary(mosek.streamtype.msg)

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

                if solsta in [mosek.solsta.optimal]:
                    expret = 0.0
                    x = [0.] * n
                    task.getxxslice(mosek.soltype.itr,
                                    offsetx + 0, offsetx + n, x)
                    for j in range(0, n):
                        expret += mu[j] * x[j]

                    stddev = [0.]
                    task.getxxslice(mosek.soltype.itr,
                                    offsets + 0, offsets + 1, stddev)

                    print("alpha = {0:.2e} exp. ret. = {1:.3e}, variance {2:.3e}".format(alpha,
↪ expret, stddev[0]))
                else:
                    print("An error occurred when solving for alpha=%e\n" % alpha)
```

11.1.3 Factor model and efficiency

In practice it is often important to solve the portfolio problem very quickly. Therefore, in this section we discuss how to improve computational efficiency at the modeling stage.

The computational cost is of course to some extent dependent on the number of constraints and variables in the optimization problem. However, in practice a more important factor is the sparsity: the number of nonzeros used to represent the problem. Indeed it is often better to focus on the number of nonzeros in G see (11.2) and try to reduce that number by for instance changing the choice of G.

In other words if the computational efficiency should be improved then it is always good idea to start with focusing at the covariance matrix. As an example assume that

$$\Sigma = D + VV^T$$

where D is a positive definite diagonal matrix. Moreover, V is a matrix with n rows and p columns. Such a model for the covariance matrix is called a factor model and usually p is much smaller than n. In practice p tends to be a small number independent of n, say less than 100.

One possible choice for G is the Cholesky factorization of Σ which requires storage proportional to $n(n+1)/2$. However, another choice is

$$G^T = \left[\begin{array}{c} D^{1/2} \\ V^T \end{array} \right]$$

because then

$$GG^T = D + VV^T.$$

This choice requires storage proportional to $n + pn$ which is much less than for the Cholesky choice of G. Indeed assuming p is a constant storage requirements are reduced by a factor of n.

The example above exploits the so-called factor structure and demonstrates that an alternative choice of G may lead to a significant reduction in the amount of storage used to represent the problem. This will in most cases also lead to a significant reduction in the solution time.

The lesson to be learned is that it is important to investigate how the covariance matrix is formed. Given this knowledge it might be possible to make a special choice for G that helps reducing the storage requirements and enhance the computational efficiency. More details about this process can be found in *[And13]*.

11.1.4 Slippage Cost

The basic Markowitz model assumes that there are no costs associated with trading the assets and that the returns of the assets are independent of the amount traded. Neither of those assumptions is usually valid in practice. Therefore, a more realistic model is

$$\begin{array}{lrcl}\text{maximize} & \mu^T x & & \\ \text{subject to} & e^T x + \sum_{j=1}^n C_j |x_j - x_j^0| & = & w + e^T x^0, \\ & x^T \Sigma x & \leq & \gamma^2, \\ & x & \geq & 0,\end{array} \tag{11.11}$$

where the function

$$C_j |x_j - x_j^0|$$

specifies the transaction costs when the holding of asset j is changed from its initial value. In the next two sections we show two different variants of this problem with two nonlinear cost functions T.

11.1.5 Market Impact Costs

If the initial wealth is fairly small and no short selling is allowed, then the holdings will be small and the traded amount of each asset must also be small. Therefore, it is reasonable to assume that the prices of the assets are independent of the amount traded. However, if a large volume of an asset is sold or purchased, the price, and hence return, can be expected to change. This effect is called market impact costs. It is common to assume that the market impact cost for asset j can be modeled by

$$C_j = m_j \sqrt{|x_j - x_j^0|}$$

where m_j is a constant that is estimated in some way by the trader. See *[GK00]* [p. 452] for details. Hence, we have

$$C_j(x_j - x_j^0) = m_j |x_j - x_j^0| \sqrt{|x_j - x_j^0|} = m_j |x_j - x_j^0|^{3/2}.$$

From the Modeling Cookbook we know that $c \geq z^{3/2}$ can be modeled directly using the power cone $\mathcal{P}_3^{2/3,1/3}$:

$$\{(c, z) : c \geq z^{3/2}, z \geq 0\} = \{(c, z) : (c, 1, z) \in \mathcal{P}_3^{2/3,1/3}\}$$

Hence, it follows that we can write the model as

$$\begin{array}{rl} z_j & = |x_j - x_j^0|, \\ (c_j, 1, z_j) & \in \mathcal{P}_3^{2/3,1/3}, \\ \sum_{j=1}^n C_j |x_j - x_j^0| & = \sum_{j=1}^n c_j m_j. \end{array}$$

Unfortunately this set of constraints is nonconvex due to the constraint

$$z_j = |x_j - x_j^0| \tag{11.12}$$

but in many cases the constraint may be replaced by the relaxed constraint

$$z_j \geq |x_j - x_j^0|, \tag{11.13}$$

which is equivalent to

$$\begin{array}{l} z_j \geq x_j - x_j^0, \\ z_j \geq -(x_j - x_j^0). \end{array} \tag{11.14}$$

For instance if the universe of assets contains a risk free asset then

$$z_j > |x_j - x_j^0| \tag{11.15}$$

cannot hold for an optimal solution.

If the optimal solution has the property (11.15) then the market impact cost within the model is larger than the true market impact cost and hence money are essentially considered garbage and removed by generating transaction costs. This may happen if a portfolio with very small risk is requested because the only way to obtain a small risk is to get rid of some of the assets by generating transaction costs. We generally assume that this is not the case and hence the models (11.12) and (11.13) are equivalent.

The above observations lead to

$$\begin{array}{lrcll}
\text{maximize} & \mu^T x & & & \\
\text{subject to} & e^T x + m^T c & = & w + e^T x^0, & \\
& [\gamma; G^T x] & \in & \mathcal{Q}^{n+1}, & \\
& z_j & \geq & x_j - x_j^0, & j = 1, \ldots, n, \\
& z_j & \geq & x_j^0 - x_j, & j = 1, \ldots, n, \\
& (c_j, 1, z_j) & \in & \mathcal{P}_3^{2/3,1/3}, & j = 1, \ldots, n, \\
& x & \geq & 0. &
\end{array} \tag{11.16}$$

The revised budget constraint

$$e^T x + m^T c = w + e^T x^0$$

specifies that the initial wealth covers the investment and the transaction costs. It should be mentioned that transaction costs of the form

$$t_j \geq z_j^p$$

where $p > 1$ is a real number can be modeled with the power cone as

$$(t_j, 1, z_j) \in \mathcal{P}_3^{1/p, 1-1/p}.$$

See Modeling Cookbook for details.

Creating a matrix formulation

One more reformulation of (11.16) is needed to bring it to the standard form (11.4).

$$\begin{array}{lrcll}
\text{maximize} & \mu^T x & & & \\
\text{subject to} & e^T x + m^T c & = & w + e^T x^0, & \\
& G^T x - t & = & 0, & \\
& z_j - x_j & \geq & -x_j^0, & j = 1, \ldots, n, \\
& z_j + x_j & \geq & x_j^0, & j = 1, \ldots, n, \\
& (s, t) & \in & \mathcal{Q}^{n+1}, & \\
& (c_j, f_j, z_j) & \in & \mathcal{P}_3^{2/3,1/3}, & j = 1, \ldots, n, \\
& x & \geq & 0, & \\
& f_j & = & 1, & j = 1, \ldots, n, \\
& s & = & \gamma, &
\end{array} \tag{11.17}$$

where $f \in \mathbb{R}^n$ is an additional variable representing the unused coordinate in the power cone. The formulation (11.17) is not the most compact possible, but it is easy to implement. **MOSEK** presolve will automatically simplify it.

The first step in developing the implementation is to chose an ordering of the variables. We will choose the following ordering:

$$\hat{x} = [x; s; t; c; z; f]$$

Table 11.2 shows the mapping between the $\hat{x}$ vector and the model variables.

Table 11.2: Storage layout for $\hat{x}$

Variable	Length	Offset
x	n	0
s	1	n
t	n	$n+1$
c	n	$2n+1$
z	n	$3n+1$
f	n	$4n+1$

The next step is to consider how the linear constraint matrix A and the remaining data vectors are laid out. Reusing the idea in Sec. 11.1.1 we can write the data in block matrix form and read off all the required coordinates. This extension of the code setting up the constraint $G^T x - t = 0$ from Sec. 11.1.1 is shown below.

Source code example

The example code in Listing 11.6 demonstrates how to implement the model (11.17).

Listing 11.6: Code implementing model (11.17).

```
import mosek

def streamprinter(text):
    print("%s" % text),

if __name__ == '__main__':

    n = 3
    gamma = 0.05
    mu = [0.1073, 0.0737, 0.0627]
    GT = [[0.1667, 0.0232, 0.0013],
          [0.0000, 0.1033, -0.0022],
          [0.0000, 0.0000, 0.0338]]
    x0 = [0.0, 0.0, 0.0]
    w = 1.0
    m = [0.01, 0.01, 0.01]

    # This value has no significance.
    inf = 0.0

    with mosek.Env() as env:
        with env.Task(0, 0) as task:
            task.set_Stream(mosek.streamtype.log, streamprinter)

            rtemp = w
            for j in range(0, n):
                rtemp += x0[j]

            # Constraints.
            task.appendcons(1 + 3 * n)
            task.putconbound(0, mosek.boundkey.fx, rtemp, rtemp)
            task.putconname(0, "budget")

            task.putconboundlist(range(1 + 0, 1 + n), n *
                                 [mosek.boundkey.fx], n * [0.0], n * [0.0])
            for j in range(1, 1 + n):
                task.putconname(j, "GT[%d]" % j)

            task.putconboundlist(range(
                1 + n, 1 + 2 * n), n * [mosek.boundkey.lo], [-x0[j] for j in range(0, n)], n *␣
↪[inf])
            for i in range(0, n):
                task.putconname(1 + n + i, "zabs1[%d]" % (1 + i))

            task.putconboundlist(range(1 + 2 * n, 1 + 3 * n),
                                 n * [mosek.boundkey.lo], x0, n * [inf])
            for i in range(0, n):
                task.putconname(1 + 2 * n + i, "zabs2[%d]" % (1 + i))

            # Offset of variables into the API variable.
            offsetx = 0
```

(continues on next page)

(continued from previous page)

```
        offsets = n
        offsett = n + 1
        offsetc = 2 * n + 1
        offsetz = 3 * n + 1
        offsetf = 4 * n + 1

        # Variables.
        task.appendvars(1 + 5 * n)

        # x variables.
        task.putclist(range(offsetx + 0, offsetx + n), mu)
        task.putaijlist(
            n * [0], range(offsetx + 0, offsetx + n), n * [1.0])
        for j in range(0, n):
            task.putaijlist(
                n * [1 + j], range(offsetx + 0, offsetx + n), GT[j])
            task.putaij(1 + n + j, offsetx + j, -1.0)
            task.putaij(1 + 2 * n + j, offsetx + j, 1.0)

        task.putvarboundlist(
            range(offsetx + 0, offsetx + n), n * [mosek.boundkey.lo], n * [0.0], n * [inf])
        for j in range(0, n):
            task.putvarname(offsetx + j, "x[%d]" % (1 + j))

        # s variable.
        task.putvarbound(offsets + 0, mosek.boundkey.fx, gamma, gamma)
        task.putvarname(offsets + 0, "s")

        # t variables.
        task.putaijlist(range(1, n + 1), range(offsett +
                                               0, offsett + n), n * [-1.0])
        task.putvarboundlist(range(offsett + 0, offsett + n),
                             n * [mosek.boundkey.fr], n * [-inf], n * [inf])
        for j in range(0, n):
            task.putvarname(offsett + j, "t[%d]" % (1 + j))

        # c variables.
        task.putaijlist(n * [0], range(offsetc, offsetc + n), m)
        task.putvarboundlist(range(offsetc, offsetc + n),
                             n * [mosek.boundkey.fr], n * [-inf], n * [inf])
        for j in range(0, n):
            task.putvarname(offsetc + j, "c[%d]" % (1 + j))

        # z variables.
        task.putaijlist(range(1 + 1 * n, 1 + 2 * n),
                        range(offsetz, offsetz + n), n * [1.0])
        task.putaijlist(range(1 + 2 * n, 1 + 3 * n),
                        range(offsetz, offsetz + n), n * [1.0])
        task.putvarboundlist(range(offsetz, offsetz + n),
                             n * [mosek.boundkey.fr], n * [-inf], n * [inf])
        for j in range(0, n):
            task.putvarname(offsetz + j, "z[%d]" % (1 + j))

        # f variables.
        task.putvarboundlist(range(offsetf, offsetf + n),
                             n * [mosek.boundkey.fx], n * [1.0], n * [1.0])
        for j in range(0, n):
            task.putvarname(offsetf + j, "f[%d]" % (1 + j))

        # quadratic cone
        task.appendcone(mosek.conetype.quad, 0.0, [
```

(continues on next page)

(continued from previous page)

```
                        offsets] + list(range(offsett, offsett + n)))
        task.putconename(0, "stddev")

        # power cones
        for k in range(0, n):
            task.appendcone(mosek.conetype.ppow, 2.0/3.0,
                            [offsetc + k, offsetf + k, offsetz + k])
            task.putconename(1 + k, "trans[%d]" % (1 + k))


        task.putobjsense(mosek.objsense.maximize)

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

        # Dump the problem to a human readable OPF file.
        task.writedata("dump.opf")

        task.optimize()

        # Display the solution summary for quick inspection of results.
        task.solutionsummary(mosek.streamtype.msg)

        expret = 0.0
        x = [0.] * n
        task.getxxslice(mosek.soltype.itr, offsetx + 0, offsetx + n, x)
        for j in range(0, n):
            expret += mu[j] * x[j]

        stddev = [0.]
        task.getxxslice(mosek.soltype.itr, offsets +
                        0, offsets + 1, stddev)

        print("\nExpected return %e for gamma %e\n" % (expret, stddev[0]))
```

The example code above produces the result

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

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

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

Listing 11.7: OPF file for problem (11.17).

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

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

(continues on next page)

(continued from previous page)

```
  [hint NUMCONE] 4 [/hint]
[/hints]

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

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

[constraints]
  [con 'budget']  'x[1]' + 'x[2]' + 'x[3]' + 1e-02 'c[1]' + 1e-02 'c[2]'
     + 1e-02 'c[3]' = 1e+00 [/con]
  [con 'GT[1]']  1.667e-01 'x[1]' + 2.32e-02 'x[2]' + 1.3e-03 'x[3]' - 't[1]' = 0e+00 [/con]
  [con 'GT[2]']  1.033e-01 'x[2]' - 2.2e-03 'x[3]' - 't[2]' = 0e+00 [/con]
  [con 'GT[3]']  3.38e-02 'x[3]' - 't[3]' = 0e+00 [/con]
  [con 'zabs1[1]'] 0e+00 <= - 'x[1]' + 'z[1]' [/con]
  [con 'zabs1[2]'] 0e+00 <= - 'x[2]' + 'z[2]' [/con]
  [con 'zabs1[3]'] 0e+00 <= - 'x[3]' + 'z[3]' [/con]
  [con 'zabs2[1]'] 0e+00 <= 'x[1]' + 'z[1]' [/con]
  [con 'zabs2[2]'] 0e+00 <= 'x[2]' + 'z[2]' [/con]
  [con 'zabs2[3]'] 0e+00 <= 'x[3]' + 'z[3]' [/con]
[/constraints]

[bounds]
  [b] 0e+00      <= 'x[1]','x[2]','x[3]' [/b]
  [b]               s =  5e-02 [/b]
  [b]               't[1]','t[2]','t[3]','c[1]','c[2]','c[3]' free [/b]
  [b]               'z[1]','z[2]','z[3]' free [/b]
  [b]               'f[1]','f[2]','f[3]' =  1e+00 [/b]
  [cone quad 'stddev'] s, 't[1]', 't[2]', 't[3]' [/cone]
  [cone ppow '6.666666666666666e-01' 'trans[1]'] 'c[1]', 'f[1]', 'z[1]' [/cone]
  [cone ppow '6.666666666666666e-01' 'trans[2]'] 'c[2]', 'f[2]', 'z[2]' [/cone]
  [cone ppow '6.666666666666666e-01' 'trans[3]'] 'c[3]', 'f[3]', 'z[3]' [/cone]
[/bounds]
```

The file verifies that the correct problem has been set up.

11.1.6 Transaction Costs

Now assume there is a cost associated with trading asset j given by

$$T_j(\Delta x_j) = \begin{cases} 0, & \Delta x_j = 0, \\ f_j + g_j|\Delta x_j|, & \text{otherwise.} \end{cases}$$

Here Δx_j is the change in the holding of asset j i.e.

$$\Delta x_j = x_j - x_j^0.$$

Hence, whenever asset j is traded we pay a fixed setup cost f_j and a variable cost of g_j per unit traded. This sort of cost function can be modeled using mixed-integer optimization, in particular using a binary variable y_j to indicate if asset j is traded. Given the assumptions about transaction costs in this section problem (11.11) may be formulated as

$$
\begin{array}{lrcll}
\text{maximize} & \mu^T x & & & \\
\text{subject to} & e^T x + f^T y + g^T z & = & w + e^T x^0, & \\
& [\gamma; G^T x] & \in & \mathcal{Q}^{n+1}, & \\
& z_j & \geq & x_j - x_j^0, & j = 1, \ldots, n, \\
& z_j & \geq & x_j^0 - x_j, & j = 1, \ldots, n, \\
& z_j & \leq & U_j y_j, & j = 1, \ldots, n, \\
& y_j & \in & \{0, 1\}, & j = 1, \ldots, n, \\
& x & \geq & 0. &
\end{array} \tag{11.18}
$$

First observe that

$$z_j \geq |x_j - x_j^0| = |\Delta x_j|.$$

Here U_j is some a priori chosen upper bound on the amount of trading in asset j and therefore if $z_j > 0$ then $y_j = 1$ has to be the case. This implies that the transaction cost for asset j is given by

$$f_j y_j + g_j z_j.$$

In our problem a safe bound for each U_j is the total initial wealth $w + e^T x^0$, however knowing a tighter bound may lead to shorter solution times.

Creating a matrix formulation

One more reformulation of (11.18) is needed to bring it to the standard form (11.4).

$$
\begin{array}{lccll}
\text{maximize} & \mu^T x & & & \\
\text{subject to} & e^T x + f^T y + g^T z & = & w + e^T x^0, & \\
& G^T x - t & = & 0, & \\
& z_j - x_j & \geq & -x_j^0, & j = 1, \ldots, n, \\
& z_j + x_j & \geq & x_j^0, & j = 1, \ldots, n, \\
& (s, t) & \in & \mathcal{Q}^{n+1}, & \\
& z_j - U_j y_j & \leq & 0, & j = 1, \ldots, n, \\
& x & \geq & 0, & \\
& y_j & \in & [0, 1], & j = 1, \ldots, n, \\
& y_j & \in & \mathbb{Z}, & j = 1, \ldots, n, \\
& s & = & \gamma. &
\end{array} \tag{11.19}
$$

We will choose the following ordering of variables:

$$\hat{x} = [x; s; t; z; y]$$

Table 11.3 shows the mapping between the $\hat{x}$ vector and the model variables.

Table 11.3: Storage layout for $\hat{x}$

Variable	Length	Offset
x	n	0
s	1	n
t	n	$n+1$
z	n	$2n+1$
y	n	$3n+1$

The next step is to consider how the linear constraint matrix A and the remaining data vectors are laid out. Reusing the idea in Sec. 11.1.1 we can write the data in block matrix form and read off all the required coordinates. This extension of the code setting up the constraint $G^T x - t = 0$ from Sec. 11.1.1 is shown below.

Example code

The following example code demonstrates how to compute an optimal portfolio when transaction setup costs are included. Note that we are now solving a problem with integer variables, and therefore the solution must be retrieved from *soltype.itg* rather than *soltype.itr*.

Listing 11.8: Code solving problem (11.18).

```
import mosek

def streamprinter(text):
    print("%s" % text),

if __name__ == '__main__':

    n = 3
    gamma = 0.05
    mu = [0.1073, 0.0737, 0.0627]
    GT = [[0.1667, 0.0232, 0.0013],
          [0.0000, 0.1033, -0.0022],
          [0.0000, 0.0000, 0.0338]]
    x0 = [0.0, 0.0, 0.0]
    w = 1.0
    f = [0.01, 0.01, 0.01]
    g = [0.001, 0.001, 0.001]

    # This value has no significance.
    inf = 0.0

    with mosek.Env() as env:
        with env.Task(0, 0) as task:
            task.set_Stream(mosek.streamtype.log, streamprinter)

            # Total wealth
            U = w + sum(x0)

            # Constraints.
            task.appendcons(1 + 4 * n)
            task.putconbound(0, mosek.boundkey.fx, U, U)
            task.putconname(0, "budget")

            task.putconboundlist(range(1 + 0, 1 + n), n *
                                 [mosek.boundkey.fx], n * [0.0], n * [0.0])
            for j in range(1, 1 + n):
                task.putconname(j, "GT[%d]" % j)

            task.putconboundlist(range(
                1 + n, 1 + 2 * n), n * [mosek.boundkey.lo], [-x0[j] for j in range(0, n)], n *␣
↪[inf])
            for i in range(0, n):
                task.putconname(1 + n + i, "zabs1[%d]" % (1 + i))

            task.putconboundlist(range(1 + 2 * n, 1 + 3 * n),
                                 n * [mosek.boundkey.lo], x0, n * [inf])
            for i in range(0, n):
                task.putconname(1 + 2 * n + i, "zabs2[%d]" % (1 + i))

            task.putconboundlist(range(1 + 3 * n, 1 + 4 * n),
                                 n * [mosek.boundkey.up], n * [-inf], n * [0.0])
            for i in range(0, n):
                task.putconname(1 + 3 * n + i, "ind[%d]" % (1 + i))
```

(continues on next page)

(continued from previous page)

```
        # Offset of variables into the API variable.
        offsetx = 0
        offsets = n
        offsett = n + 1
        offsetz = 2 * n + 1
        offsety = 3 * n + 1

        # Variables.
        task.appendvars(1 + 4 * n)

        # x variables.
        task.putclist(range(offsetx + 0, offsetx + n), mu)
        task.putaijlist(
            n * [0], range(offsetx + 0, offsetx + n), n * [1.0])
        for j in range(0, n):
            task.putaijlist(
                n * [1 + j], range(offsetx + 0, offsetx + n), GT[j])
            task.putaij(1 + n + j, offsetx + j, -1.0)
            task.putaij(1 + 2 * n + j, offsetx + j, 1.0)

        task.putvarboundlist(
            range(offsetx + 0, offsetx + n), n * [mosek.boundkey.lo], n * [0.0], n * [inf])
        for j in range(0, n):
            task.putvarname(offsetx + j, "x[%d]" % (1 + j))

        # s variable.
        task.putvarbound(offsets + 0, mosek.boundkey.fx, gamma, gamma)
        task.putvarname(offsets + 0, "s")

        # t variables.
        task.putaijlist(range(1, n + 1), range(offsett +
                                               0, offsett + n), n * [-1.0])
        task.putvarboundlist(range(offsett + 0, offsett + n),
                             n * [mosek.boundkey.fr], n * [-inf], n * [inf])
        for j in range(0, n):
            task.putvarname(offsett + j, "t[%d]" % (1 + j))

        # z variables.
        task.putaijlist(n * [0], range(offsetz, offsetz + n), g)
        task.putaijlist(range(1 + 1 * n, 1 + 2 * n),
                        range(offsetz, offsetz + n), n * [1.0])
        task.putaijlist(range(1 + 2 * n, 1 + 3 * n),
                        range(offsetz, offsetz + n), n * [1.0])
        task.putaijlist(range(1 + 3 * n, 1 + 4 * n),
                        range(offsetz, offsetz + n), n * [1.0])
        task.putvarboundlist(range(offsetz, offsetz + n),
                             n * [mosek.boundkey.fr], n * [-inf], n * [inf])
        for j in range(0, n):
            task.putvarname(offsetz + j, "z[%d]" % (1 + j))

        # y variables.
        task.putaijlist(n * [0], range(offsety, offsety + n), f)
        task.putaijlist(range(1 + 3 * n, 1 + 4 * n),
                        range(offsety, offsety + n), n * [-U])
        task.putvarboundlist(range(offsety, offsety + n),
                             n * [mosek.boundkey.ra], n * [0.0], n * [1.0])
        task.putvartypelist(range(offsety, offsety + n), n * [mosek.variabletype.type_int])
        for j in range(0, n):
            task.putvarname(offsety + j, "y[%d]" % (1 + j))

        # quadratic cone
```

(continues on next page)

(continued from previous page)

```
    task.appendcone(mosek.conetype.quad, 0.0, [
                    offsets] + list(range(offsett, offsett + n)))
    task.putconename(0, "stddev")

    task.putobjsense(mosek.objsense.maximize)

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

    # Dump the problem to a human readable OPF file.
    # task.writedata("dump.opf")

    task.optimize()

    # Display the solution summary for quick inspection of results.
    task.solutionsummary(mosek.streamtype.msg)

    expret = 0.0
    x = [0.] * n
    task.getxxslice(mosek.soltype.itg, offsetx + 0, offsetx + n, x)
    for j in range(0, n):
        expret += mu[j] * x[j]

    stddev = [0.]
    task.getxxslice(mosek.soltype.itg, offsets +
                    0, offsets + 1, stddev)

    print("\nExpected return %e for gamma %e\n" % (expret, stddev[0]))
```

11.1.7 Cardinality constraints

Another method to reduce costs involved with processing transactions is to only change positions in a small number of assets. In other words, at most k of the differences $|\Delta x_j| = |x_j - x_j^0|$ are allowed to be non-zero, where k is (much) smaller than the total number of assets n.

This type of constraint can be again modeled by introducing a binary variable y_j which indicates if $\Delta x_j \neq 0$ and bounding the sum of y_j. The basic Markowitz model then gets updated as follows:

$$
\begin{array}{lrcll}
\text{maximize} & \mu^T x & & & \\
\text{subject to} & e^T x & = & w + e^T x^0, & \\
 & [\gamma; G^T x] & \in & \mathcal{Q}^{n+1}, & \\
 & U_j y_j & \geq & |x_j - x_j^0|, & j = 1, \ldots, n, \\
 & y_j & \in & \{0, 1\}, & j = 1, \ldots, n, \\
 & e^T y & \leq & k, & \\
 & x & \geq & 0, &
\end{array}
\tag{11.20}
$$

were U_j is some a priori chosen upper bound on the amount of trading in asset j. This guarantees that $|x_j - x_j^0|$ forces $y_j = 1$ and therefore $e^T y$ counts the number of assets in which we trade. In our problem a safe bound for each U_j is the total initial wealth $w + e^T x^0$, however knowing a tighter bound may lead to shorter solution times.

Creating a matrix formulation

One more reformulation of (11.20) is needed to bring it to the standard form (11.4).

$$
\begin{array}{lrcll}
\text{maximize} & \mu^T x & & & \\
\text{subject to} & e^T x & = & w + e^T x^0, & \\
 & G^T x - t & = & 0, & \\
 & z_j - x_j & \geq & -x_j^0, & j = 1,\ldots,n, \\
 & z_j + x_j & \geq & x_j^0, & j = 1,\ldots,n, \\
 & (s,t) & \in & \mathcal{Q}^{n+1}, & \\
 & z_j - U_j y_j & \leq & 0, & j = 1,\ldots,n, \\
 & e^T y & \leq & k, & \\
 & x & \geq & 0, & \\
 & y_j & \in & [0,1], & j = 1,\ldots,n, \\
 & y_j & \in & \mathbb{Z}, & j = 1,\ldots,n, \\
 & s & = & \gamma. &
\end{array}
\tag{11.21}
$$

We will choose the following ordering of variables:

$$\hat{x} = [x; s; t; z; y]$$

Table 11.4 shows the mapping between the $\hat{x}$ vector and the model variables.

Table 11.4: Storage layout for $\hat{x}$

Variable	Length	Offset
x	n	0
s	1	n
t	n	$n+1$
z	n	$2n+1$
y	n	$3n+1$

The next step is to consider how the linear constraint matrix A and the remaining data vectors are laid out. Reusing the idea in Sec. 11.1.1 we can write the data in block matrix form and read off all the required coordinates. This extension of the code setting up the constraint $G^T x - t = 0$ from Sec. 11.1.1 is shown below.

Example code

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

Listing 11.9: Code solving problem (11.20).

```
def markowitz_with_card(n, x0, w, gamma, mu, GT, k):
    with mosek.Env() as env:
        with env.Task(0, 0) as task:
            task.set_Stream(mosek.streamtype.log, streamprinter)

            # Total wealth
            U = w + sum(x0)

            # Constraints.
            task.appendcons(2 + 4 * n)
            task.putconbound(0, mosek.boundkey.fx, U, U)
            task.putconname(0, "budget")

            task.putconbound(1 + 4 * n, mosek.boundkey.up, -inf, k)
            task.putconname(0, "cardinality")
```

(continues on next page)

(continued from previous page)

```
            task.putconboundlist(range(1 + 0, 1 + n), n *
                                 [mosek.boundkey.fx], n * [0.0], n * [0.0])
            for j in range(1, 1 + n):
                task.putconname(j, "GT[%d]" % j)

            task.putconboundlist(range(
                1 + n, 1 + 2 * n), n * [mosek.boundkey.lo], [-x0[j] for j in range(0, n)], n *␣
↪[inf])
            for i in range(0, n):
                task.putconname(1 + n + i, "zabs1[%d]" % (1 + i))

            task.putconboundlist(range(1 + 2 * n, 1 + 3 * n),
                                 n * [mosek.boundkey.lo], x0, n * [inf])
            for i in range(0, n):
                task.putconname(1 + 2 * n + i, "zabs2[%d]" % (1 + i))

            task.putconboundlist(range(1 + 3 * n, 1 + 4 * n),
                                 n * [mosek.boundkey.up], n * [-inf], n * [0.0])
            for i in range(0, n):
                task.putconname(1 + 3 * n + i, "ind[%d]" % (1 + i))

            # Offset of variables into the API variable.
            offsetx = 0
            offsets = n
            offsett = n + 1
            offsetz = 2 * n + 1
            offsety = 3 * n + 1

            # Variables.
            task.appendvars(1 + 4 * n)

            # x variables.
            task.putclist(range(offsetx + 0, offsetx + n), mu)
            task.putaijlist(n * [0], range(offsetx + 0, offsetx + n), n * [1.0])
            for j in range(0, n):
                task.putaijlist(n * [1 + j], range(offsetx + 0, offsetx + n), GT[j])
                task.putaij(1 + n + j, offsetx + j, -1.0)
                task.putaij(1 + 2 * n + j, offsetx + j, 1.0)

            task.putvarboundlist(
                range(offsetx + 0, offsetx + n), n * [mosek.boundkey.lo], n * [0.0], n * [inf])
            for j in range(0, n):
                task.putvarname(offsetx + j, "x[%d]" % (1 + j))

            # s variable.
            task.putvarbound(offsets + 0, mosek.boundkey.fx, gamma, gamma)
            task.putvarname(offsets + 0, "s")

            # t variables.
            task.putaijlist(range(1, n + 1), range(offsett +
                                                   0, offsett + n), n * [-1.0])
            task.putvarboundlist(range(offsett + 0, offsett + n),
                                 n * [mosek.boundkey.fr], n * [-inf], n * [inf])
            for j in range(0, n):
                task.putvarname(offsett + j, "t[%d]" % (1 + j))

            # z variables.
            task.putaijlist(range(1 + 1 * n, 1 + 2 * n),
                            range(offsetz, offsetz + n), n * [1.0])
            task.putaijlist(range(1 + 2 * n, 1 + 3 * n),
                            range(offsetz, offsetz + n), n * [1.0])
```

(continues on next page)

(continued from previous page)

```
        task.putaijlist(range(1 + 3 * n, 1 + 4 * n),
                        range(offsetz, offsetz + n), n * [1.0])
        task.putvarboundlist(range(offsetz, offsetz + n),
                             n * [mosek.boundkey.fr], n * [-inf], n * [inf])
        for j in range(0, n):
            task.putvarname(offsetz + j, "z[%d]" % (1 + j))

        # y variables.
        task.putaijlist(n * [1 + 4 * n], range(offsety, offsety + n), n * [1.0])
        task.putaijlist(range(1 + 3 * n, 1 + 4 * n),
                        range(offsety, offsety + n), n * [-U])
        task.putvarboundlist(range(offsety, offsety + n),
                             n * [mosek.boundkey.ra], n * [0.0], n * [1.0])
        task.putvartypelist(range(offsety, offsety + n), n * [mosek.variabletype.type_int])
        for j in range(0, n):
            task.putvarname(offsety + j, "y[%d]" % (1 + j))

        # quadratic cone
        task.appendcone(mosek.conetype.quad, 0.0, [
                        offsets] + list(range(offsett, offsett + n)))
        task.putconename(0, "stddev")

        task.putobjsense(mosek.objsense.maximize)

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

        # Dump the problem to a human readable OPF file.
        # task.writedata("dump.opf")

        task.optimize()

        # Display the solution summary for quick inspection of results.
        task.solutionsummary(mosek.streamtype.msg)

        xx = [0.] * n
        task.getxxslice(mosek.soltype.itg, offsetx + 0, offsetx + n, xx)
        return xx
```

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

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

11.2 Logistic regression

Logistic regression is an example of a binary classifier, where the output takes one two values 0 or 1 for each data point. We call the two values *classes*.

Formulation as an optimization problem

Define the sigmoid function

$$S(x) = \frac{1}{1 + \exp(-x)}.$$

Next, given an observation $x \in \mathbb{R}^d$ and a weights $\theta \in \mathbb{R}^d$ we set

$$h_\theta(x) = S(\theta^T x) = \frac{1}{1 + \exp(-\theta^T x)}.$$

The weights vector θ is part of the setup of the classifier. The expression $h_\theta(x)$ is interpreted as the probability that x belongs to class 1. When asked to classify x the returned answer is

$$x \mapsto \begin{cases} 1 & h_\theta(x) \geq 1/2, \\ 0 & h_\theta(x) < 1/2. \end{cases}$$

When training a logistic regression algorithm we are given a sequence of training examples x_i, each labelled with its class $y_i \in \{0,1\}$ and we seek to find the weights θ which maximize the likelihood function

$$\prod_i h_\theta(x_i)^{y_i}(1-h_\theta(x_i))^{1-y_i}.$$

Of course every single y_i equals 0 or 1, so just one factor appears in the product for each training data point. By taking logarithms we can define the logistic loss function:

$$J(\theta) = -\sum_{i:y_i=1} \log(h_\theta(x_i)) - \sum_{i:y_i=0} \log(1-h_\theta(x_i)).$$

The training problem with regularization (a standard technique to prevent overfitting) is now equivalent to

$$\min_\theta J(\theta) + \lambda\|\theta\|_2.$$

This can equivalently be phrased as

$$\begin{array}{lllll} \text{minimize} & \sum_i t_i + \lambda r & & & \\ \text{subject to} & t_i & \geq -\log(h_\theta(x)) & = \log(1+\exp(-\theta^T x_i)) & \text{if } y_i = 1, \\ & t_i & \geq -\log(1-h_\theta(x)) & = \log(1+\exp(\theta^T x_i)) & \text{if } y_i = 0, \\ & r & \geq \|\theta\|_2. & & \end{array} \tag{11.22}$$

Implementation

As can be seen from (11.22) the key point is to implement the softplus bound $t \geq \log(1+e^u)$, which is the simplest example of a log-sum-exp constraint for two scalar variables t, u. This is equivalent to

$$\exp(u-t) + \exp(-t) \leq 1$$

and further to

$$\begin{array}{rcll} (z_1, 1, u-t) & \in & K_{\exp} & (z_1 \geq \exp(u-t)), \\ (z_2, 1, -t) & \in & K_{\exp} & (z_2 \geq \exp(-t)), \\ z_1 + z_2 & \leq & 1. & \end{array} \tag{11.23}$$

To feed these constraints into **MOSEK** we add more auxiliary variables q_1, q_2, v_1, v_2 with constraints $(z_1, q_1, v_1) \in K_{\exp}$, $(z_2, q_2, v_2) \in K_{\exp}$, $q_1 = q_2 = 1$, $v_1 = u - t$ and $v_2 = -t$.

Listing 11.10: Implementation of $t \geq \log(1+e^u)$ as in (11.23).

```
# t_i >= log( 1 + exp(u_i) ), i = 0..n-1
# Adds auxiliary variables and constraints
def softplus(task, t, u, n):
    nvar = task.getnumvar()
    ncon = task.getnumcon()
    task.appendvars(6*n)
    task.appendcons(3*n)
    z1, z2, v1, v2, q1, q2 = nvar, nvar+n, nvar+2*n, nvar+3*n, nvar+4*n, nvar+5*n
    zcon, v1con, v2con = ncon, ncon+n, ncon+2*n

    # z1 + z2 = 1
```

(continues on next page)

(continued from previous page)

```
    task.putaijlist(range(zcon, zcon+n), range(z1, z1+n), [1]*n)
    task.putaijlist(range(zcon, zcon+n), range(z2, z2+n), [1]*n)
    # u - t - v1 = 0
    task.putaijlist(range(v1con, v1con+n), range(u, u+n), [1]*n)
    task.putaijlist(range(v1con, v1con+n), range(t, t+n), [-1]*n)
    task.putaijlist(range(v1con, v1con+n), range(v1, v1+n), [-1]*n)
    # - t - v2 = 0
    task.putaijlist(range(v2con, v2con+n), range(t, t+n), [-1]*n)
    task.putaijlist(range(v2con, v2con+n), range(v2, v2+n), [-1]*n)
    # Bounds for all constraints
    task.putconboundslice(ncon, ncon+3*n, [boundkey.fx]*(3*n), [1]*n+[0]*(2*n), 
↪[1]*n+[0]*(2*n))
    # Bounds for variables
    task.putvarboundsliceconst(nvar, nvar+4*n, boundkey.fr, -inf, inf)
    task.putvarboundsliceconst(nvar+4*n, nvar+6*n, boundkey.fx, 1, 1)

    # Cones
    for i in range(n):
        task.appendcone(conetype.pexp, 0.0, [z1+i, q1+i, v1+i])
        task.appendcone(conetype.pexp, 0.0, [z2+i, q2+i, v2+i])
```

Once we have this subroutine, it is easy to implement a function that builds the regularized loss function model (11.22).

Listing 11.11: Implementation of (11.22).

```
# Model logistic regression (regularized with full 2-norm of theta)
# X - n x d matrix of data points
# y - length n vector classifying training points
# lamb - regularization parameter
def logisticRegression(env, X, y, lamb=1.0):
    n, d = int(X.shape[0]), int(X.shape[1])         # num samples, dimension

    with env.Task() as task:
        # Variables [r; theta; t; u]
        nvar = 1+d+2*n
        task.appendvars(nvar)
        task.putvarboundsliceconst(0, nvar, boundkey.fr, -inf, inf)
        r, theta, t, u, = 0, 1, 1+d, 1+d+n

        # Constraints: theta'*X +/- u = 0
        task.appendcons(n)
        task.putconboundsliceconst(0, n, boundkey.fx, 0.0, 0.0)

        # Objective lambda*r + sum(t)
        task.putcj(r, lamb)
        task.putclist(range(t, t+n), [1.0]*n)

        # The X block in theta'*X +/- u = 0
        uCoeff = []
        for i in range(n):
            task.putaijlist([i]*d, range(theta, theta+d), X[i])
            uCoeff.append(1 if y[i] == 1 else -1)
        # +/- coefficients in u depending on y
        task.putaijlist(range(n), range(u, u+n), uCoeff)

        # Softplus function constraints
        softplus(task, t, u, n)

        # Regularization
```

(continues on next page)

(continued from previous page)

```
        task.appendconeseq(conetype.quad, 0.0, 1+d, r)

        # Solution
        task.optimize()
        xx = [0.0]*d
        task.getxxslice(soltype.itr, theta, theta+d, xx)

        return xx
```

Example: 2D dataset fitting

In the next figure we apply logistic regression to the training set of 2D points taken from the example ex2data2.txt . The two-dimensional dataset was converted into a feature vector $x \in \mathbb{R}^{28}$ using monomial coordinates of degrees at most 6.

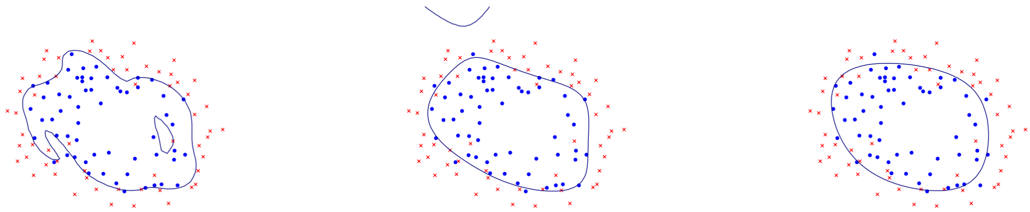


Fig. 11.1: Logistic regression example with none, medium and strong regularization (small, medium, large λ). Without regularization we get obvious overfitting.

11.3 Concurrent optimizer

The idea of the concurrent optimizer is to run multiple optimizations of **the same problem** simultaneously, and pick the one that provides the fastest or best answer. This approach is especially useful for problems which require a very long time and it is hard to say in advance which optimizer or algorithm will perform best.

The major applications of concurrent optimization we describe in this section are:

- Using the interior-point and simplex optimizers simultaneously on a linear problem. Note that any solution present in the task will also be used for hot-starting the simplex algorithms. One possible scenario would therefore be running a hot-start simplex in parallel with interior point, taking advantage of both the stability of the interior-point method and the ability of the simplex method to use an initial solution.
- Using multiple instances of the mixed-integer optimizer to solve many copies of one mixed-integer problem. This is not in contradiction with the run-to-run determinism of **MOSEK** if a different value of the MIO seed parameter *iparam.mio_seed* is set in each instance. As a result each setting leads to a different optimizer run (each of them being deterministic in its own right).

The downloadable file contains usage examples of both kinds.

11.3.1 Common setup

We first define a method that runs a number of optimization tasks in parallel, using the standard multithreading setup available in the language. All tasks register for a callback function which will signal them to interrupt as soon as the first task completes successfully (with response code *rescode.ok*).

Listing 11.12: Simple callback function which signals the optimizer to stop.

```
# Defines a Mosek callback function whose only function
# is to indicate if the optimizer should be stopped.
stop = False
firstStop = -1
def cbFun(code):
  return 1 if stop else 0
```

When all remaining tasks respond to the stop signal, response codes and statuses are returned to the caller, together with the index of the task which won the race.

Listing 11.13: A routine for parallel task race.

```
def runTask(num, task, res, trm):
  global stop
  global firstStop
  try:
    trm[num] = task.optimize();
    res[num] = mosek.rescode.ok
  except mosek.MosekException as e:
    trm[num] = mosek.rescode.err_unknown
    res[num] = e.errno
  finally:
    # If this finished with success, inform other tasks to interrupt
    if res[num] == mosek.rescode.ok:
      if not stop:
        firstStop = num
      stop = True

def optimize(tasks):
  n = len(tasks)
  res = [ mosek.rescode.err_unknown ] * n
  trm = [ mosek.rescode.err_unknown ] * n

  # Set a callback function
  for t in tasks:
    t.set_Progress(cbFun)

  # Start parallel optimizations, one per task
  jobs = [ Thread(target=runTask, args=(i, tasks[i], res, trm)) for i in range(n) ]
  for j in jobs:
    j.start()
  for j in jobs:
    j.join()

  # For debugging, print res and trm codes for all optimizers
  for i in range(n):
    print("Optimizer  {0}   res {1}   trm {2}".format(i, res[i], trm[i]))

  return firstStop, res, trm
```

11.3.2 Linear optimization

We use the multithreaded setup to run the interior-point and simplex optimizers simultaneously on a linear problem. The next methods simply clones the given task and sets a different optimizer for each. The result is the clone which finished first.

Listing 11.14: Concurrent optimization with different optimizers.

```
def optimizeconcurrent(task, optimizers):
  n = len(optimizers)
  tasks = [ mosek.Task(task) for _ in range(n) ]

  # Choose various optimizers for cloned tasks
  for i in range(n):
    tasks[i].putintparam(mosek.iparam.optimizer, optimizers[i])

  # Solve tasks in parallel
  firstOK, res, trm = optimize(tasks)

  if firstOK >= 0:
    return firstOK, tasks[firstOK], trm[firstOK], res[firstOK]
  else:
    return -1, None, None, None
```

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

Listing 11.15: Calling concurrent linear optimization.

```
        optimizers = [
          mosek.optimizertype.conic,
          mosek.optimizertype.dual_simplex,
          mosek.optimizertype.primal_simplex
        ]

        idx, t, trm, res = optimizeconcurrent(task, optimizers)
```

11.3.3 Mixed-integer optimization

We use the multithreaded setup to run many, differently seeded copies of the mixed-integer optimizer. This approach is most useful for hard problems where we don't expect an optimal solution in reasonable time. The input task would typically contain a time limit. It is possible that all the cloned tasks reach the time limit, in which case it doesn't really mater which one terminated first. Instead we examine all the task clones for the best objective value.

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

```
def optimizeconcurrentMIO(task, seeds):
  n = len(seeds)
  tasks = [ mosek.Task(task) for _ in range(n) ]

  # Choose various seeds for cloned tasks
  for i in range(n):
    tasks[i].putintparam(mosek.iparam.mio_seed, seeds[i])

  # Solve tasks in parallel
  firstOK, res, trm = optimize(tasks)

  if firstOK >= 0:
    # Pick the task that ended with res = ok
    # and contains an integer solution with best objective value
    sense = task.getobjsense();
    bestObj = 1.0e+10 if sense == mosek.objsense.minimize else -1.0e+10
    bestPos = -1

    for i in range(n):
      print("{0}   {1}".format(i,tasks[i].getprimalobj(mosek.soltype.itg)))
```

(continues on next page)

(continued from previous page)

```
  for i in range(n):
    if ((res[i] == mosek.rescode.ok) and
        (tasks[i].getsolsta(mosek.soltype.itg) == mosek.solsta.prim_feas or
         tasks[i].getsolsta(mosek.soltype.itg) == mosek.solsta.integer_optimal) and
       ((tasks[i].getprimalobj(mosek.soltype.itg) < bestObj)
         if (sense == mosek.objsense.minimize) else
        (tasks[i].getprimalobj(mosek.soltype.itg) > bestObj))):
      bestObj = tasks[i].getprimalobj(mosek.soltype.itg)
      bestPos = i

  if bestPos >= 0:
    return bestPos, tasks[bestPos], trm[bestPos], res[bestPos]

return -1, None, None, None
```

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

Listing 11.17: Calling concurrent integer optimization.

```
      seeds = [ 42, 13, 71749373 ]

      idx, t, trm, res = optimizeconcurrentMIO(task, seeds)
```

Chapter 12

Problem Formulation and Solutions

In this chapter we will discuss the following issues:

- The formal, mathematical formulations of the problem types that **MOSEK** can solve and their duals.
- The solution information produced by **MOSEK**.
- The infeasibility certificate produced by **MOSEK** if the problem is infeasible.

For the underlying mathematical concepts, derivations and proofs see the Modeling Cookbook or any book on convex optimization. This chapter explains how the related data is organized specifically within the **MOSEK** API.

12.1 Linear Optimization

MOSEK accepts linear optimization problems of the form

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

where

- m is the number of constraints.
- n is the number of decision variables.
- $x \in \mathbb{R}^n$ is a vector of decision variables.
- $c \in \mathbb{R}^n$ is the linear part of the objective function.
- $c^f \in \mathbb{R}$ is a constant term in the objective
- $A \in \mathbb{R}^{m \times n}$ is the constraint matrix.
- $l^c \in \mathbb{R}^m$ is the lower limit on the activity for the constraints.
- $u^c \in \mathbb{R}^m$ is the upper limit on the activity for the constraints.
- $l^x \in \mathbb{R}^n$ is the lower limit on the activity for the variables.
- $u^x \in \mathbb{R}^n$ is the upper limit on the activity for the variables.

Lower and upper bounds can be infinite, or in other words the corresponding bound may be omitted.

A primal solution (x) is *(primal) feasible* if it satisfies all constraints in (12.1). If (12.1) has at least one primal feasible solution, then (12.1) is said to be (primal) feasible. In case (12.1) does not have a feasible solution, the problem is said to be *(primal) infeasible*

12.1.1 Duality for Linear Optimization

Corresponding to the primal problem (12.1), there is a dual problem

$$\begin{array}{lccl} \text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x + c^f & & \\ & A^T y + s_l^x - s_u^x & = & c, \\ \text{subject to} & -y + s_l^c - s_u^c & = & 0, \\ & s_l^c, s_u^c, s_l^x, s_u^x & \geq & 0. \end{array} \tag{12.2}$$

If a bound in the primal problem is plus or minus infinity, the corresponding dual variable is fixed at 0, and we use the convention that the product of the bound value and the corresponding dual variable is 0. This is equivalent to removing variable $(s_l^x)_j$ from the dual problem. In other words:

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

A solution

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

to the dual problem is feasible if it satisfies all the constraints in (12.2). If (12.2) has at least one feasible solution, then (12.2) is *(dual) feasible*, otherwise the problem is *(dual) infeasible*.

A solution

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

is denoted a *primal-dual feasible solution*, if (x^*) is a solution to the primal problem (12.1) and $(y^*, (s_l^c)^*, (s_u^c)^*, (s_l^x)^*, (s_u^x)^*)$ is a solution to the corresponding dual problem (12.2). We also define an auxiliary vector

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

containing the activities of linear constraints.

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

$$\begin{array}{l} c^T x^* + c^f - \left\{ (l^c)^T (s_l^c)^* - (u^c)^T (s_u^c)^* + (l^x)^T (s_l^x)^* - (u^x)^T (s_u^x)^* + c^f \right\} \\ = \sum_{i=0}^{m-1} [(s_l^c)_i^* ((x_i^c)^* - l_i^c) + (s_u^c)_i^* (u_i^c - (x_i^c)^*)] \\ + \sum_{j=0}^{n-1} \left[(s_l^x)_j^* (x_j - l_j^x) + (s_u^x)_j^* (u_j^x - x_j^*) \right] \geq 0 \end{array} \tag{12.3}$$

where the first relation can be obtained by transposing and multiplying the dual constraints (12.2) by x^* and $(x^c)^*$ respectively, and the second relation comes from the fact that each term in each sum is nonnegative. It follows that the primal objective will always be greater than or equal to the dual objective.

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

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

are satisfied.

If (12.1) has an optimal solution and **MOSEK** solves the problem successfully, both the primal and dual solution are reported, including a status indicating the exact state of the solution.

12.1.2 Infeasibility for Linear Optimization

Primal Infeasible Problems

If the problem (12.1) is infeasible (has no feasible solution), **MOSEK** will report a certificate of primal infeasibility: The dual solution reported is the certificate of infeasibility, and the primal solution is undefined.

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

$$
\begin{array}{ll}
\text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x \\
\text{subject to} & \\
& A^T y + s_l^x - s_u^x = 0, \\
& -y + s_l^c - s_u^c = 0, \\
& s_l^c, s_u^c, s_l^x, s_u^x \geq 0,
\end{array}
\tag{12.4}
$$

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

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

to (12.4) so that

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

Such a solution implies that (12.4) is unbounded, and that (12.1) is infeasible.

Dual Infeasible Problems

If the problem (12.2) is infeasible (has no feasible solution), **MOSEK** will report a certificate of dual infeasibility: The primal solution reported is the certificate of infeasibility, and the dual solution is undefined.

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

$$
\begin{array}{lccccc}
\text{minimize} & & & c^T x & & \\
\text{subject to} & \hat{l}^c & \leq & Ax & \leq & \hat{u}^c, \\
& \hat{l}^x & \leq & x & \leq & \hat{u}^x,
\end{array}
\tag{12.5}
$$

where

$$
\hat{l}_i^c = \left\{ \begin{array}{ll} 0 & \text{if } l_i^c > -\infty, \\ -\infty & \text{otherwise,} \end{array} \right\} \quad \text{and} \quad \hat{u}_i^c := \left\{ \begin{array}{ll} 0 & \text{if } u_i^c < \infty, \\ \infty & \text{otherwise,} \end{array} \right\}
$$

and

$$
\hat{l}_j^x = \left\{ \begin{array}{ll} 0 & \text{if } l_j^x > -\infty, \\ -\infty & \text{otherwise,} \end{array} \right\} \quad \text{and} \quad \hat{u}_j^x := \left\{ \begin{array}{ll} 0 & \text{if } u_j^x < \infty, \\ \infty & \text{otherwise,} \end{array} \right\}
$$

such that

$$
c^T x < 0.
$$

Such a solution implies that (12.5) is unbounded, and that (12.2) is infeasible.

In case that both the primal problem (12.1) and the dual problem (12.2) are infeasible, **MOSEK** will report only one of the two possible certificates — which one is not defined (**MOSEK** returns the first certificate found).

12.1.3 Minimalization vs. Maximalization

When the objective sense of problem (12.1) is maximization, i.e.

$$
\begin{array}{lccccc}
\text{maximize} & & & c^T x + c^f & & \\
\text{subject to} & l^c & \leq & Ax & \leq & u^c, \\
& l^x & \leq & x & \leq & u^x,
\end{array}
$$

the objective sense of the dual problem changes to minimization, and the domain of all dual variables changes sign in comparison to (12.2). The dual problem thus takes the form

$$
\begin{array}{ll}
\text{minimize} & (l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x + c^f \\
\text{subject to} & \\
& A^T y + s_l^x - s_u^x = c, \\
& -y + s_l^c - s_u^c = 0, \\
& s_l^c, s_u^c, s_l^x, s_u^x \leq 0.
\end{array}
$$

This means that the duality gap, defined in (12.3) as the primal minus the dual objective value, becomes nonpositive. It follows that the dual objective will always be greater than or equal to the primal objective. The primal infeasibility certificate will be reported by **MOSEK** as a solution to the system

$$\begin{array}{r}A^T y + s_l^x - s_u^x = 0,\\ -y + s_l^c - s_u^c = 0,\\ s_l^c, s_u^c, s_l^x, s_u^x \leq 0,\end{array} \tag{12.6}$$

such that the objective value is strictly negative

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

Similarly, the certificate of dual infeasibility is an x satisfying the requirements of (12.5) such that $c^T x > 0$.

12.2 Conic Optimization

Conic optimization is an extension of linear optimization (see Sec. 12.1) allowing conic domains to be specified for subsets of the problem variables. A conic optimization problem to be solved by **MOSEK** can be written as

$$\begin{array}{lccccc}\text{minimize} & & & c^T x + c^f & & \\ \text{subject to} & l^c & \leq & Ax & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x, \\ & & & x \in \mathcal{K}, & & \end{array} \tag{12.7}$$

where

- m is the number of constraints.
- n is the number of decision variables.
- $x \in \mathbb{R}^n$ is a vector of decision variables.
- $c \in \mathbb{R}^n$ is the linear part of the objective function.
- $c^f \in \mathbb{R}$ is a constant term in the objective
- $A \in \mathbb{R}^{m \times n}$ is the constraint matrix.
- $l^c \in \mathbb{R}^m$ is the lower limit on the activity for the constraints.
- $u^c \in \mathbb{R}^m$ is the upper limit on the activity for the constraints.
- $l^x \in \mathbb{R}^n$ is the lower limit on the activity for the variables.
- $u^x \in \mathbb{R}^n$ is the upper limit on the activity for the variables.

Lower and upper bounds can be infinite, or in other words the corresponding bound may be omitted.

The set $\mathcal{K}$ is a Cartesian product of convex cones, namely $\mathcal{K} = \mathcal{K}_1 \times \cdots \times \mathcal{K}_p$. Having the domain restriction $x \in \mathcal{K}$, is thus equivalent to

$$x^t \in \mathcal{K}_t \subseteq \mathbb{R}^{n_t},$$

where $x = (x^1, \ldots, x^p)$ is a partition of the problem variables. Please note that the n-dimensional Euclidean space $\mathbb{R}^n$ is a cone itself, so simple linear variables are still allowed. The user only needs to specify subsets of variables which belong to non-trivial cones.

In this section we discuss the formulations which apply to the following cones supported by **MOSEK**:

- The set $\mathbb{R}^n$.
- The zero cone $\{(0, \ldots, 0)\}$.
- Quadratic cone

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

- Rotated quadratic cone

$$\mathcal{Q}_r^n = \left\{ x \in \mathbb{R}^n : 2x_1 x_2 \geq \sum_{j=3}^{n} x_j^2, \quad x_1 \geq 0, \quad x_2 \geq 0 \right\}.$$

- Primal exponential cone

$$K_{\exp} = \left\{ x \in \mathbb{R}^3 : x_1 \geq x_2 \exp(x_3/x_2), \quad x_1, x_2 \geq 0 \right\}$$

as well as its dual

$$K_{\exp}^* = \left\{ x \in \mathbb{R}^3 : x_1 \geq -x_3 e^{-1} \exp(x_2/x_3), \quad x_3 \leq 0, x_1 \geq 0 \right\}.$$

- Primal power cone (with parameter $0 < \alpha < 1$)

$$\mathcal{P}_n^{\alpha,1-\alpha} = \left\{ x \in \mathbb{R}^n : x_1^{\alpha} x_2^{1-\alpha} \geq \sqrt{\sum_{j=3}^{n} x_j^2}, \quad x_1, x_2 \geq 0 \right\}$$

as well as its dual

$$(\mathcal{P}_n^{\alpha,1-\alpha})^* = \left\{ x \in \mathbb{R}^n : \left(\frac{x_1}{\alpha}\right)^{\alpha} \left(\frac{x_2}{1-\alpha}\right)^{1-\alpha} \geq \sqrt{\sum_{j=3}^{n} x_j^2}, \quad x_1, x_2 \geq 0 \right\}.$$

MOSEK supports also the cone of positive semidefinite matrices. Since that is handled through a separate interface, we discuss it in Sec. 12.3.

12.2.1 Duality for Conic Optimization

Corresponding to the primal problem (12.7), there is a dual problem

$$\begin{array}{ll} \text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x + c^f \\ \text{subject to} & \\ & A^T y + s_l^x - s_u^x + s_n^x = c \\ & -y + s_l^c - s_u^c = 0, \\ & s_l^c, s_u^c, s_l^x, s_u^x \geq 0, \\ & s_n^x \in \mathcal{K}^*, \end{array} \tag{12.8}$$

where the dual cone $\mathcal{K}^*$ is a Cartesian product of the cones dual to $\mathcal{K}_t$. In practice this means that s_n^x has one entry for each entry in x. Please note that the dual problem of the dual problem is identical to the original primal problem.

If a bound in the primal problem is plus or minus infinity, the corresponding dual variable is fixed at 0, and we use the convention that the product of the bound value and the corresponding dual variable is 0. This is equivalent to removing variable $(s_l^x)_j$ from the dual problem. In other words:

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

A solution

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

to the dual problem is feasible if it satisfies all the constraints in (12.8). If (12.8) has at least one feasible solution, then (12.8) is *(dual) feasible*, otherwise the problem is *(dual) infeasible*.

A solution

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

is denoted a *primal-dual feasible solution*, if (x^*) is a solution to the primal problem (12.7) and $(y^*, (s_l^c)^*, (s_u^c)^*, (s_l^x)^*, (s_u^x)^*, (s_n^x)^*)$ is a solution to the corresponding dual problem (12.8). We also define an auxiliary vector

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

containing the activities of linear constraints.

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

$$\begin{array}{l} c^T x^* + c^f - \left\{ (l^c)^T (s_l^c)^* - (u^c)^T (s_u^c)^* + (l^x)^T (s_l^x)^* - (u^x)^T (s_u^x)^* + c^f \right\} \\ = \sum_{i=0}^{m-1} \left[(s_l^c)_i^* ((x_i^c)^* - l_i^c) + (s_u^c)_i^* (u_i^c - (x_i^c)^*) \right] \\ + \sum_{j=0}^{n-1} \left[(s_l^x)_j^* (x_j - l_j^x) + (s_u^x)_j^* (u_j^x - x_j^*) \right] + \sum_{j=0}^{n-1} (s_n^x)_j^* x_j^* \geq 0 \end{array} \tag{12.9}$$

where the first relation can be obtained by transposing and multiplying the dual constraints (12.2) by x^* and $(x^c)^*$ respectively, and the second relation comes from the fact that each term in each sum is nonnegative. It follows that the primal objective will always be greater than or equal to the dual objective.

It is well-known that, under some non-degeneracy assumptions that exclude ill-posed cases, a conic optimization problem has an optimal solution if and only if there exist feasible primal-dual solution so that the duality gap is zero, or, equivalently, that the *complementarity conditions*

$$\begin{array}{rcll} (s_l^c)_i^* ((x_i^c)^* - l_i^c) & = & 0, & i = 0, \ldots, m-1, \\ (s_u^c)_i^* (u_i^c - (x_i^c)^*) & = & 0, & i = 0, \ldots, m-1, \\ (s_l^x)_j^* (x_j^* - l_j^x) & = & 0, & j = 0, \ldots, n-1, \\ (s_u^x)_j^* (u_j^x - x_j^*) & = & 0, & j = 0, \ldots, n-1, \\ \sum_{j=0}^{n-1} (s_n^x)_j^* x_j^* & = & 0. & \end{array} \tag{12.10}$$

are satisfied.

If (12.7) has an optimal solution and **MOSEK** solves the problem successfully, both the primal and dual solution are reported, including a status indicating the exact state of the solution.

12.2.2 Infeasibility for Conic Optimization

Primal Infeasible Problems

If the problem (12.7) is infeasible (has no feasible solution), **MOSEK** will report a certificate of primal infeasibility: The dual solution reported is the certificate of infeasibility, and the primal solution is undefined.

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

$$\begin{array}{ll} \text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x \\ \text{subject to} & \\ & A^T y + s_l^x - s_u^x + s_n^x = 0, \\ & -y + s_l^c - s_u^c = 0, \\ & s_l^c, s_u^c, s_l^x, s_u^x \geq 0, \\ & s_n^x \in \mathcal{K}^*, \end{array} \tag{12.11}$$

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

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

to (12.11) so that

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

Such a solution implies that (12.11) is unbounded, and that (12.7) is infeasible.

Dual Infeasible Problems

If the problem (12.8) is infeasible (has no feasible solution), **MOSEK** will report a certificate of dual infeasibility: The primal solution reported is the certificate of infeasibility, and the dual solution is undefined.

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

$$\begin{array}{lccccc} \text{minimize} & & & c^T x & & \\ \text{subject to} & \hat{l}^c & \leq & Ax & \leq & \hat{u}^c, \\ & \hat{l}^x & \leq & x & \leq & \hat{u}^x, \\ & & & x \in K, & & \end{array} \tag{12.12}$$

where

$$\hat{l}_i^c = \left\{ \begin{array}{ll} 0 & \text{if } l_i^c > -\infty, \\ -\infty & \text{otherwise,} \end{array} \right\} \quad \text{and} \quad \hat{u}_i^c := \left\{ \begin{array}{ll} 0 & \text{if } u_i^c < \infty, \\ \infty & \text{otherwise,} \end{array} \right\} \tag{12.13}$$

and

$$\hat{l}_j^x = \left\{ \begin{array}{ll} 0 & \text{if } l_j^x > -\infty, \\ -\infty & \text{otherwise,} \end{array} \right\} \quad \text{and} \quad \hat{u}_j^x := \left\{ \begin{array}{ll} 0 & \text{if } u_j^x < \infty, \\ \infty & \text{otherwise,} \end{array} \right\} \tag{12.14}$$

such that

$$c^T x < 0.$$

Such a solution implies that (12.12) is unbounded, and that (12.8) is infeasible.

In case that both the primal problem (12.7) and the dual problem (12.8) are infeasible, **MOSEK** will report only one of the two possible certificates — which one is not defined (**MOSEK** returns the first certificate found).

12.2.3 Minimalization vs. Maximalization

When the objective sense of problem (12.7) is maximization, i.e.

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

the objective sense of the dual problem changes to minimization, and the domain of all dual variables changes sign in comparison to (12.2). The dual problem thus takes the form

$$\begin{array}{ll} \text{minimize} & (l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x + c^f \\ \text{subject to} & A^T y + s_l^x - s_u^x + s_n^x = c, \\ & -y + s_l^c - s_u^c = 0, \\ & s_l^c, s_u^c, s_l^x, s_u^x \leq 0, \\ & -s_n^x \in \mathcal{K}^* \end{array}$$

This means that the duality gap, defined in (12.9) as the primal minus the dual objective value, becomes nonpositive. It follows that the dual objective will always be greater than or equal to the primal objective. The primal infeasibility certificate will be reported by **MOSEK** as a solution to the system

$$\begin{array}{c} A^T y + s_l^x - s_u^x + s_n^x = 0, \\ -y + s_l^c - s_u^c = 0, \\ s_l^c, s_u^c, s_l^x, s_u^x \leq 0, \\ -s_n^x \in \mathcal{K}^* \end{array} \tag{12.15}$$

such that the objective value is strictly negative

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

Similarly, the certificate of dual infeasibility is an x satisfying the requirements of (12.12) such that $c^T x > 0$.

12.3 Semidefinite Optimization

Semidefinite optimization is an extension of conic optimization (see Sec. 12.2) allowing positive semidefinite matrix variables to be used in addition to the usual scalar variables. All the other parts of the input are defined exactly as in Sec. 12.2, and the discussion from that section applies verbatim to all properties of problems with semidefinite variables. We only briefly indicate how the corresponding formulae should be modified with semidefinite terms.

A semidefinite optimization problem can be written as

$$\begin{array}{lccccl}
\text{minimize} & & & \sum_{j=0}^{n-1} c_j x_j + \sum_{j=0}^{p-1} \left\langle \overline{C}_j, \overline{X}_j \right\rangle + c^f & & \\
\text{subject to} & l_i^c & \leq & \sum_{j=0}^{n-1} a_{ij} x_j + \sum_{j=0}^{p-1} \left\langle \overline{A}_{ij}, \overline{X}_j \right\rangle & \leq & u_i^c, \quad i = 0, \ldots, m-1 \\
 & l_j^x & \leq & x_j & \leq & u_j^x, \quad j = 0, \ldots, n-1 \\
 & & & x \in \mathcal{K}, & & \\
 & & & \overline{X}_j \in \mathcal{S}_+^{r_j}, & & \quad j = 0, \ldots, p-1
\end{array} \tag{12.16}$$

where the problem has p symmetric positive semidefinite variables $\overline{X}_j \in \mathcal{S}_+^{r_j}$ of dimension r_j with symmetric coefficient matrices $\overline{C}_j \in \mathcal{S}^{r_j}$ and $\overline{A}_{i,j} \in \mathcal{S}^{r_j}$. We use standard notation for the matrix inner product, i.e., for $U, V \in \mathbb{R}^{m \times n}$ we have

$$\langle U, V \rangle := \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} U_{ij} V_{ij}.$$

As always we write $A = (a_{i,j})$ for the linear coefficient matrix.

Duality

The definition of the dual problem (12.8) becomes:

$$\begin{array}{ll}
\text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x + c^f \\
\text{subject to} & \\
 & A^T y + s_l^x - s_u^x + s_n^x = c \\
 & -y + s_l^c - s_u^c = 0, \\
 & \overline{C}_j - \sum_{i=0}^{m-1} y_i \overline{A}_{ij} = \overline{S}_j, \qquad j = 0, \ldots, p-1 \\
 & s_l^c, s_u^c, s_l^x, s_u^x \geq 0, \\
 & s_n^x \in \mathcal{K}^*, \\
 & \overline{S}_j \in \mathcal{S}_+^{r_j}, \qquad j = 0, \ldots, p-1.
\end{array} \tag{12.17}$$

The duality gap (12.9) is computed as:

$$\begin{array}{l}
c^T x^* + c^f - \left\{ (l^c)^T (s_l^c)^* - (u^c)^T (s_u^c)^* + (l^x)^T (s_l^x)^* - (u^x)^T (s_u^x)^* + c^f \right\} \\
= \sum_{i=0}^{m-1} [(s_l^c)_i^* ((x_i^c)^* - l_i^c) + (s_u^c)_i^* (u_i^c - (x_i^c)^*)] \\
+ \sum_{j=0}^{n-1} \left[(s_l^x)_j^* (x_j - l_j^x) + (s_u^x)_j^* (u_j^x - x_j^*) \right] + \sum_{j=0}^{n-1} (s_n^x)_j^* x_j^* + \sum_{j=0}^{p-1} \langle \overline{X}_j, \overline{S}_j \rangle \geq 0.
\end{array} \tag{12.18}$$

Complementarity conditions (12.10) include the additional relation:

$$\langle \overline{X}_j, \overline{S}_j \rangle = 0 \quad j = 0, \ldots, p-1. \tag{12.19}$$

Infeasibility

A certificate of primal infeasibility (12.11) is now a feasible solution to:

$$\begin{array}{ll}
\text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x \\
\text{subject to} & \\
 & A^T y + s_l^x - s_u^x + s_n^x = 0, \\
 & -y + s_l^c - s_u^c = 0, \\
 & \sum_{i=0}^{m-1} y_i \overline{A}_{ij} + \overline{S}_j = 0, \qquad j = 0, \ldots, p-1 \\
 & s_l^c, s_u^c, s_l^x, s_u^x \geq 0, \\
 & s_n^x \in \mathcal{K}^*, \\
 & \overline{S}_j \in \mathcal{S}_+^{r_j}, \qquad j = 0, \ldots, p-1.
\end{array} \tag{12.20}$$

such that the objective value is strictly positive.

Similarly, a dual infeasibility certificate (12.12) is a feasible solution to

$$\begin{array}{lccccll}
\text{minimize} & & & \sum_{j=0}^{n-1} c_j x_j + \sum_{j=0}^{p-1} \langle \overline{C}_j, \overline{X}_j \rangle & & & \\
\text{subject to} & \hat{l}_i^c & \leq & \sum_{j=0}^{n-1} a_{ij} x_j + \sum_{j=0}^{p-1} \langle \overline{A}_{ij}, \overline{X}_j \rangle & \leq & \hat{u}_i^c, & i = 0, \ldots, m-1 \\
 & \hat{l}_j^x & \leq & x_j & \leq & \hat{u}_j^x, & j = 0, \ldots, n-1 \\
 & & & x \in \mathcal{K}, & & & \\
 & & & \overline{X}_j \in \mathcal{S}_+^{r_j}, & & & j = 0, \ldots, p-1
\end{array} \tag{12.21}$$

where the modified bounds are as in (12.13) and (12.14) and the objective value is strictly negative.

12.4 Quadratic and Quadratically Constrained Optimization

A convex quadratic and quadratically constrained optimization problem has the form

$$\begin{array}{lccccll}
\text{minimize} & & & \frac{1}{2} x^T Q^o x + c^T x + c^f & & & \\
\text{subject to} & l_k^c & \leq & \frac{1}{2} x^T Q^k x + \sum_{j=0}^{n-1} a_{kj} x_j & \leq & u_k^c, & k = 0, \ldots, m-1, \\
 & l_j^x & \leq & x_j & \leq & u_j^x, & j = 0, \ldots, n-1,
\end{array} \tag{12.22}$$

where all variables and bounds have the same meaning as for linear problems (see Sec. 12.1) and Q^o and all Q^k are symmetric matrices. Moreover, for convexity, Q^o must be a positive semidefinite matrix and Q^k must satisfy

$$\begin{array}{rcl}
-\infty < l_k^c & \Rightarrow & Q^k \text{ is negative semidefinite,} \\
u_k^c < \infty & \Rightarrow & Q^k \text{ is positive semidefinite,} \\
-\infty < l_k^c \leq u_k^c < \infty & \Rightarrow & Q^k = 0.
\end{array}$$

The convexity requirement is very important and **MOSEK** checks whether it is fulfilled.

12.4.1 A Recommendation

Any convex quadratic optimization problem can be reformulated as a conic quadratic optimization problem, see Modeling Cookbook and *[And13]*. In fact **MOSEK** does such conversion internally as a part of the solution process for the following reasons:

- the conic optimizer is numerically more robust than the one for quadratic problems.
- the conic optimizer is usually faster because quadratic cones are simpler than quadratic functions, even though the conic reformulation usually has more constraints and variables than the original quadratic formulation.
- it is easy to dualize the conic formulation if deemed worthwhile potentially leading to (huge) computational savings.

However, instead of relying on the automatic reformulation we recommend to formulate the problem as a conic problem from scratch because:

- it saves the computational overhead of the reformulation including the convexity check. A conic problem is convex by construction and hence no convexity check is needed for conic problems.
- usually the modeler can do a better reformulation than the automatic method because the modeler can exploit the knowledge of the problem at hand.

To summarize we recommend to formulate quadratic problems and in particular quadratically constrained problems directly in conic form.

12.4.2 Duality for Quadratic and Quadratically Constrained Optimization

The dual problem corresponding to the quadratic and quadratically constrained optimization problem (12.22) is given by

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

The dual problem is related to the dual problem for linear optimization (see Sec. 12.1.1), but depends on the variable x which in general can not be eliminated. In the solutions reported by **MOSEK**, the value of x is the same for the primal problem (12.22) and the dual problem (12.23).

12.4.3 Infeasibility for Quadratic Optimization

In case **MOSEK** finds a problem to be infeasible it reports a certificate of infeasibility. We write them out explicitly for quadratic problems, that is when $Q^k = 0$ for all k and quadratic terms appear only in the objective Q^o. In this case the constraints both in the primal and dual problem are linear, and **MOSEK** produces for them the same infeasibility certificate as for linear problems.

The certificate of primal infeasibility is a solution to the problem (12.4) such that the objective value is strictly positive.

The certificate of dual infeasibility is a solution to the problem (12.5) together with an additional constraint

$$
Q^o x = 0
$$

such that the objective value is strictly negative.

Chapter 13

Optimizers

The most essential part of **MOSEK** are the optimizers:

- *primal simplex* (linear problems),
- *dual simplex* (linear problems),
- *interior-point* (linear, quadratic and conic problems),
- *mixed-integer* (problems with integer variables).

The structure of a successful optimization process is roughly:

- **Presolve**
  1. *Elimination*: Reduce the size of the problem.
  2. *Dualizer*: Choose whether to solve the primal or the dual form of the problem.
  3. *Scaling*: Scale the problem for better numerical stability.
- **Optimization**
  1. *Optimize*: Solve the problem using selected method.
  2. *Terminate*: Stop the optimization when specific termination criteria have been met.
  3. *Report*: Return the solution or an infeasibility certificate.

The preprocessing stage is transparent to the user, but useful to know about for tuning purposes. The purpose of the preprocessing steps is to make the actual optimization more efficient and robust. We discuss the details of the above steps in the following sections.

13.1 Presolve

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

1. remove redundant constraints,
2. eliminate fixed variables,
3. remove linear dependencies,
4. substitute out (implied) free variables, and
5. reduce the size of the optimization problem in general.

After the presolved problem has been optimized the solution is automatically postsolved so that the returned solution is valid for the original problem. Hence, the presolve is completely transparent. For further details about the presolve phase, please see *[AA95]* and *[AGMX96]*.

It is possible to fine-tune the behavior of the presolve or to turn it off entirely. If presolve consumes too much time or memory compared to the reduction in problem size gained it may be disabled. This is done by setting the parameter `iparam.presolve_use` to `presolvemode.off`. The two most time-consuming steps of the presolve are

- the eliminator, and
- the linear dependency check.

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

Numerical issues in the presolve

During the presolve the problem is reformulated so that it hopefully solves faster. However, in rare cases the presolved problem may be harder to solve then the original problem. The presolve may also be infeasible although the original problem is not. If it is suspected that presolved problem is much harder to solve than the original, we suggest to first turn the eliminator off by setting the parameter *iparam.presolve_eliminator_max_num_tries* to 0. If that does not help, then trying to turn entire presolve off may help.

Since all computations are done in finite precision, the presolve employs some tolerances when concluding a variable is fixed or a constraint is redundant. If it happens that **MOSEK** incorrectly concludes a problem is primal or dual infeasible, then it is worthwhile to try to reduce the parameters *dparam.presolve_tol_x* and *dparam.presolve_tol_s*. However, if reducing the parameters actually helps then this should be taken as an indication that the problem is badly formulated.

Eliminator

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

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

y is an implied free variable that can be substituted out of the problem, if deemed worthwhile. If the eliminator consumes too much time or memory compared to the reduction in problem size gained it may be disabled. This can be done by setting the parameter *iparam.presolve_eliminator_max_num_tries* to 0. In rare cases the eliminator may cause that the problem becomes much hard to solve.

Linear dependency checker

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

$$\begin{array}{rcl} x_1 + x_2 + x_3 & = & 1, \\ x_1 + 0.5x_2 & = & 0.5, \\ 0.5x_2 + x_3 & = & 0.5. \end{array}$$

contain exactly one linear dependency. This implies that one of the constraints can be dropped without changing the set of feasible solutions. Removing linear dependencies is in general a good idea since it reduces the size of the problem. Moreover, the linear dependencies are likely to introduce numerical problems in the optimization phase. It is best practice to build models without linear dependencies, but that is not always easy for the user to control. If the linear dependencies are removed at the modeling stage, the linear dependency check can safely be disabled by setting the parameter *iparam.presolve_lindep_use* to *onoffkey.off*.

Dualizer

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

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

Note that currently only linear and conic (but not semidefinite) problems may be automatically dualized.

Scaling

Problems containing data with large and/or small coefficients, say $1.0e+9$ or $1.0e-7$, are often hard to solve. Significant digits may be truncated in calculations with finite precision, which can result in the optimizer relying on inaccurate data. Since computers work in finite precision, extreme coefficients should be avoided. In general, data around the same *order of magnitude* is preferred, and we will refer to a problem, satisfying this loose property, as being *well-scaled*. If the problem is not well scaled, **MOSEK** will try to scale (multiply) constraints and variables by suitable constants. **MOSEK** solves the scaled problem to improve the numerical properties.

The scaling process is transparent, i.e. the solution to the original problem is reported. It is important to be aware that the optimizer terminates when the termination criterion is met on the scaled problem, therefore significant primal or dual infeasibilities may occur after unscaling for badly scaled problems. The best solution of this issue is to reformulate the problem, making it better scaled.

By default **MOSEK** heuristically chooses a suitable scaling. The scaling for interior-point and simplex optimizers can be controlled with the parameters *iparam.intpnt_scaling* and *iparam.sim_scaling* respectively.

13.2 Linear Optimization

13.2.1 Optimizer Selection

Two different types of optimizers are available for linear problems: The default is an interior-point method, and the alternative is the simplex method (primal or dual). The optimizer can be selected using the parameter *iparam.optimizer*.

The Interior-point or the Simplex Optimizer?

Given a linear optimization problem, which optimizer is the best: the simplex or the interior-point optimizer? It is impossible to provide a general answer to this question. However, the interior-point optimizer behaves more predictably: it tends to use between 20 and 100 iterations, almost independently of problem size, but cannot perform warm-start. On the other hand the simplex method can take advantage of an initial solution, but is less predictable from cold-start. The interior-point optimizer is used by default.

The Primal or the Dual Simplex Variant?

MOSEK provides both a primal and a dual simplex optimizer. Predicting which simplex optimizer is faster is impossible, however, in recent years the dual optimizer has seen several algorithmic and computational improvements, which, in our experience, make it faster on average than the primal version. Still, it depends much on the problem structure and size. Setting the *iparam.optimizer* parameter to *optimizertype.free_simplex* instructs **MOSEK** to choose one of the simplex variants automatically.

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

13.2.2 The Interior-point Optimizer

The purpose of this section is to provide information about the algorithm employed in the **MOSEK** interior-point optimizer for linear problems and about its termination criteria.

The homogeneous primal-dual problem

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

$$\begin{array}{lccc} \text{minimize} & c^T x & & \\ \text{subject to} & Ax & = & b, \\ & x \geq 0. & & \end{array} \tag{13.1}$$

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

Since it is not known beforehand whether problem (13.1) has an optimal solution, is primal infeasible or is dual infeasible, the optimization algorithm must deal with all three situations. This is the reason why **MOSEK** solves the so-called homogeneous model

$$\begin{array}{rcl} Ax - b\tau & = & 0, \\ A^T y + s - c\tau & = & 0, \\ -c^T x + b^T y - \kappa & = & 0, \\ x, s, \tau, \kappa & \geq & 0, \end{array} \tag{13.2}$$

where y and s correspond to the dual variables in (13.1), and τ and κ are two additional scalar variables. Note that the homogeneous model (13.2) always has solution since

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

is a solution, although not a very interesting one. Any solution

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

to the homogeneous model (13.2) satisfies

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

Moreover, there is always a solution that has the property $\tau^* + \kappa^* > 0$.

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

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

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

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

is a primal-dual optimal solution (see Sec. 12.1 for the mathematical background on duality and optimality).

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

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

This implies that at least one of

$$c^T x^* < 0 \tag{13.3}$$

or

$$b^T y^* > 0 \tag{13.4}$$

is satisfied. If (13.3) is satisfied then x^* is a certificate of dual infeasibility, whereas if (13.4) is satisfied then y^* is a certificate of primal infeasibility.

In summary, by computing an appropriate solution to the homogeneous model, all information required for a solution to the original problem is obtained. A solution to the homogeneous model can be computed using a primal-dual interior-point algorithm *[And09]*.

Interior-point Termination Criterion

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

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

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

to homogeneous model is generated, where

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

Optimal case

Whenever the trial solution satisfies the criterion

$$\begin{aligned} \left\| A\frac{x^k}{\tau^k} - b \right\|_\infty &\leq \epsilon_p (1 + \|b\|_\infty), \\ \left\| A^T\frac{y^k}{\tau^k} + \frac{s^k}{\tau^k} - c \right\|_\infty &\leq \epsilon_d (1 + \|c\|_\infty), \text{ and} \\ \min\left(\frac{(x^k)^T s^k}{(\tau^k)^2}, \left|\frac{c^T x^k}{\tau^k} - \frac{b^T y^k}{\tau^k}\right| \right) &\leq \epsilon_g \max\left(1, \frac{\min(|c^T x^k|, |b^T y^k|)}{\tau^k}\right), \end{aligned} \tag{13.5}$$

the interior-point optimizer is terminated and

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

is reported as the primal-dual optimal solution. The interpretation of (13.5) is that the optimizer is terminated if

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

Dual infeasibility certificate

On the other hand, if the trial solution satisfies

$$-\epsilon_i c^T x^k > \frac{\|c\|_\infty}{\max(1, \|b\|_\infty)} \left\| A x^k \right\|_\infty$$

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

$$\left\| A x^k \right\|_\infty > 0,$$

and define

$$\bar{x} := \epsilon_i \frac{\max(1, \|b\|_\infty)}{\|A x^k\|_\infty \|c\|_\infty} x^k.$$

It is easy to verify that

$$\|A\bar{x}\|_\infty = \epsilon_i \frac{\max(1, \|b\|_\infty)}{\|c\|_\infty} \text{ and } -c^T \bar{x} > 1,$$

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

Primal infeasibility certificate

Finally, if

$$\epsilon_i b^T y^k > \frac{\|b\|_\infty}{\max(1, \|c\|_\infty)} \left\|A^T y^k + s^k\right\|_\infty$$

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

Adjusting optimality criteria

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

Table 13.1: Parameters employed in termination criterion

ToleranceParameter	name
ε_p	*dparam.intpnt_tol_pfeas*
ε_d	*dparam.intpnt_tol_dfeas*
ε_g	*dparam.intpnt_tol_rel_gap*
ε_i	*dparam.intpnt_tol_infeas*

The default values of the termination tolerances are chosen such that for a majority of problems appearing in practice it is not possible to achieve much better accuracy. Therefore, tightening the tolerances usually is not worthwhile. However, an inspection of (13.5) reveals that the quality of the solution depends on $\|b\|_\infty$ and $\|c\|_\infty$; the smaller the norms are, the better the solution accuracy.

The interior-point method as implemented by **MOSEK** will converge toward optimality and primal and dual feasibility at the same rate *[And09]*. This means that if the optimizer is stopped prematurely then it is very unlikely that either the primal or dual solution is feasible. Another consequence is that in most cases all the tolerances, ε_p, ε_d, ε_g and ε_i, have to be relaxed together to achieve an effect.

If the optimizer terminates without locating a solution that satisfies the termination criteria, for example because of a stall or other numerical issues, then it will check if the solution found up to that point satisfies the same criteria with all tolerances multiplied by the value of *dparam.intpnt_co_tol_near_rel* . If this is the case, the solution is still declared as optimal.

The basis identification discussed in Sec. 13.2.2 requires an optimal solution to work well; hence basis identification should be turned off if the termination criterion is relaxed.

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

Basis Identification

An interior-point optimizer does not return an optimal basic solution unless the problem has a unique primal and dual optimal solution. Therefore, the interior-point optimizer has an optional post-processing step that computes an optimal basic solution starting from the optimal interior-point solution. More information about the basis identification procedure may be found in *[AY96]*. In the following we provide an overall idea of the procedure.

There are some cases in which a basic solution could be more valuable:

- a basic solution is often more accurate than an interior-point solution,
- a basic solution can be used to warm-start the simplex algorithm in case of reoptimization,
- a basic solution is in general more sparse, i.e. more variables are fixed to zero. This is particularly appealing when solving continuous relaxations of mixed integer problems, as well as in all applications in which sparser solutions are preferred.

To illustrate how the basis identification routine works, we use the following trivial example:

$$\begin{array}{lccc} \text{minimize} & x+y & & \\ \text{subject to} & x+y & = & 1, \\ & x, y \geq 0. & & \end{array}$$

It is easy to see that all feasible solutions are also optimal. In particular, there are two basic solutions, namely

$$\begin{array}{rcl} (x_1^*, y_1^*) & = & (1,0), \\ (x_2^*, y_2^*) & = & (0,1). \end{array}$$

The interior point algorithm will actually converge to the center of the optimal set, i.e. to $(x^*, y^*) = (1/2, 1/2)$ (to see this in **MOSEK** deactivate *Presolve*).

In practice, when the algorithm gets close to the optimal solution, it is possible to construct in polynomial time an initial basis for the simplex algorithm from the current interior point solution. This basis is used to warm-start the simplex algorithm that will provide the optimal basic solution. In most cases the constructed basis is optimal, or very few iterations are required by the simplex algorithm to make it optimal and hence the final *clean-up* phase be short. However, for some cases of ill-conditioned problems the additional simplex clean up phase may take of lot a time.

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

- *iparam.intpnt_basis*,
- *iparam.bi_ignore_max_iter*, and
- *iparam.bi_ignore_num_error*

control when basis identification is performed.

The type of simplex algorithm to be used (primal/dual) can be tuned with the parameter *iparam.bi_clean_optimizer*, and the maximum number of iterations can be set with *iparam.bi_max_iterations*.

Finally, it should be mentioned that there is no guarantee on which basic solution will be returned.

The Interior-point Log

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

```
Optimizer  - threads                : 1
Optimizer  - solved problem         : the dual
Optimizer  - Constraints            : 2
Optimizer  - Cones                  : 0
Optimizer  - Scalar variables       : 6                 conic                  : 0
Optimizer  - Semi-definite variables: 0                 scalarized             : 0
Factor     - setup time             : 0.00              dense det. time        : 0.00
Factor     - ML order time          : 0.00              GP order time          : 0.00
Factor     - nonzeros before factor : 3                 after factor           : 3
Factor     - dense dim.             : 0                 flops                  : 7.00e+001
ITE PFEAS    DFEAS    GFEAS    PRSTATUS   POBJ              DOBJ              MU       TIME
0   1.0e+000 8.6e+000 6.1e+000 1.00e+000  0.000000000e+000  -2.208000000e+003 1.0e+000 0.00
1   1.1e+000 2.5e+000 1.6e-001 0.00e+000  -7.901380925e+003 -7.394611417e+003 2.5e+000 0.00
2   1.4e-001 3.4e-001 2.1e-002 8.36e-001  -8.113031650e+003 -8.055866001e+003 3.3e-001 0.00
3   2.4e-002 5.8e-002 3.6e-003 1.27e+000  -7.777530698e+003 -7.766471080e+003 5.7e-002 0.01
4   1.3e-004 3.2e-004 2.0e-005 1.08e+000  -7.668323435e+003 -7.668207177e+003 3.2e-004 0.01
5   1.3e-008 3.2e-008 2.0e-009 1.00e+000  -7.668000027e+003 -7.668000015e+003 3.2e-008 0.01
6   1.3e-012 3.2e-012 2.0e-013 1.00e+000  -7.667999994e+003 -7.667999994e+003 3.2e-012 0.01
```

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

Using the same notation as in Sec. 13.2.2 the columns of the iteration log have the following meaning:

- `ITE`: Iteration index k.
- `PFEAS`: $\left\|Ax^k - b\tau^k\right\|_\infty$. The numbers in this column should converge monotonically towards zero but may stall at low level due to rounding errors.
- `DFEAS`: $\left\|A^T y^k + s^k - c\tau^k\right\|_\infty$. The numbers in this column should converge monotonically towards zero but may stall at low level due to rounding errors.
- `GFEAS`: $|-c^T x^k + b^T y^k - \kappa^k|$. The numbers in this column should converge monotonically towards zero but may stall at low level due to rounding errors.
- `PRSTATUS`: This number converges to 1 if the problem has an optimal solution whereas it converges to -1 if that is not the case.

- POBJ: $c^T x^k/\tau^k$. An estimate for the primal objective value.
- DOBJ: $b^T y^k/\tau^k$. An estimate for the dual objective value.
- MU: $\frac{(x^k)^T s^k+\tau^k\kappa^k}{n+1}$. The numbers in this column should always converge to zero.
- TIME: Time spent since the optimization started.

13.2.3 The Simplex Optimizer

An alternative to the interior-point optimizer is the simplex optimizer. The simplex optimizer uses a different method that allows exploiting an initial guess for the optimal solution to reduce the solution time. Depending on the problem it may be faster or slower to use an initial guess; see Sec. 13.2.1 for a discussion. **MOSEK** provides both a primal and a dual variant of the simplex optimizer.

Simplex Termination Criterion

The simplex optimizer terminates when it finds an optimal basic solution or an infeasibility certificate. A basic solution is optimal when it is primal and dual feasible; see Sec. 12.1 for a definition of the primal and dual problem. Due to the fact that computations are performed in finite precision **MOSEK** allows violations of primal and dual feasibility within certain tolerances. The user can control the allowed primal and dual tolerances with the parameters *dparam.basis_tol_x* and *dparam.basis_tol_s*.

Setting the parameter *iparam.optimizer* to *optimizertype.free_simplex* instructs **MOSEK** to select automatically between the primal and the dual simplex optimizers. Hence, **MOSEK** tries to choose the best optimizer for the given problem and the available solution. The same parameter can also be used to force one of the variants.

Starting From an Existing Solution

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

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

Numerical Difficulties in the Simplex Optimizers

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

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

- Raise tolerances for allowed primal or dual feasibility: increase the value of
  - *dparam.basis_tol_x*, and
  - *dparam.basis_tol_s*.
- Raise or lower pivot tolerance: Change the *dparam.simplex_abs_tol_piv* parameter.
- Switch optimizer: Try another optimizer.
- Switch off crash: Set both *iparam.sim_primal_crash* and *iparam.sim_dual_crash* to 0.
- Experiment with other pricing strategies: Try different values for the parameters

   - *iparam.sim_primal_selection* and
   - *iparam.sim_dual_selection* .

- If you are using warm-starts, in rare cases switching off this feature may improve stability. This is controlled by the *iparam.sim_hotstart* parameter.
- Increase maximum number of set-backs allowed controlled by *iparam.sim_max_num_setbacks* .
- If the problem repeatedly becomes infeasible try switching off the special degeneracy handling. See the parameter *iparam.sim_degen* for details.

The Simplex Log

Below is a typical log output from the simplex optimizer:

```
Optimizer  - solved problem         : the primal
Optimizer  - Constraints            : 667
Optimizer  - Scalar variables       : 1424              conic                  : 0
Optimizer  - hotstart               : no
ITER      DEGITER(%)  PFEAS       DFEAS       POBJ                  DOBJ                  TIME␣
↪      TOTTIME
0         0.00        1.43e+05    NA          6.5584140832e+03      NA                    0.00␣
↪      0.02
1000      1.10        0.00e+00    NA          1.4588289726e+04      NA                    0.13␣
↪      0.14
2000      0.75        0.00e+00    NA          7.3705564855e+03      NA                    0.21␣
↪      0.22
3000      0.67        0.00e+00    NA          6.0509727712e+03      NA                    0.29␣
↪      0.31
4000      0.52        0.00e+00    NA          5.5771203906e+03      NA                    0.38␣
↪      0.39
4533      0.49        0.00e+00    NA          5.5018458883e+03      NA                    0.42␣
↪      0.44
```

The first lines summarize the problem the optimizer is solving. This is followed by the iteration log, with the following meaning:

- `ITER`: Number of iterations.
- `DEGITER(%)`: Ratio of degenerate iterations.
- `PFEAS`: Primal feasibility measure reported by the simplex optimizer. The numbers should be 0 if the problem is primal feasible (when the primal variant is used).
- `DFEAS`: Dual feasibility measure reported by the simplex optimizer. The number should be 0 if the problem is dual feasible (when the dual variant is used).
- `POBJ`: An estimate for the primal objective value (when the primal variant is used).
- `DOBJ`: An estimate for the dual objective value (when the dual variant is used).
- `TIME`: Time spent since this instance of the simplex optimizer was invoked (in seconds).
- `TOTTIME`: Time spent since optimization started (in seconds).

13.3 Conic Optimization - Interior-point optimizer

For conic optimization problems only an interior-point type optimizer is available.

13.3.1 The homogeneous primal-dual problem

The interior-point optimizer is an implementation of the so-called homogeneous and self-dual algorithm. For a detailed description of the algorithm, please see [ART03]. In order to keep our discussion simple we will assume that **MOSEK** solves a conic optimization problem of the form:

$$\begin{array}{lccc} \text{minimize} & c^T x & & \\ \text{subject to} & Ax & = & b, \\ & x \in \mathcal{K} & & \end{array} \tag{13.6}$$

where K is a convex cone. The corresponding dual problem is

$$\begin{array}{lccc} \text{maximize} & b^T y & & \\ \text{subject to} & A^T y + s & = & c, \\ & s \in \mathcal{K}^* & & \end{array} \tag{13.7}$$

where $\mathcal{K}^*$ is the dual cone of $\mathcal{K}$. See Sec. 12.2 for definitions.

Since it is not known beforehand whether problem (13.6) has an optimal solution, is primal infeasible or is dual infeasible, the optimization algorithm must deal with all three situations. This is the reason that **MOSEK** solves the so-called homogeneous model

$$\begin{array}{rcl} Ax - b\tau & = & 0, \\ A^T y + s - c\tau & = & 0, \\ -c^T x + b^T y - \kappa & = & 0, \\ x & \in & \mathcal{K}, \\ s & \in & \mathcal{K}^*, \\ \tau, \kappa & \geq & 0, \end{array} \tag{13.8}$$

where y and s correspond to the dual variables in (13.6), and τ and κ are two additional scalar variables. Note that the homogeneous model (13.8) always has a solution since

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

is a solution, although not a very interesting one. Any solution

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

to the homogeneous model (13.8) satisfies

$$(x^*)^T s^* + \tau^* \kappa^* = 0$$

i.e. complementarity. Observe that $x^* \in \mathcal{K}$ and $s^* \in \mathcal{K}^*$ implies

$$(x^*)^T s^* \geq 0$$

and therefore

$$\tau^* \kappa^* = 0.$$

since $\tau^*, \kappa^* \geq 0$. Hence, at least one of τ^* and κ^* is zero.

First, assume that $\tau^* > 0$ and hence $\kappa^* = 0$. It follows that

$$\begin{array}{rcl} A\frac{x^*}{\tau^*} & = & b, \\ A^T \frac{y^*}{\tau^*} + \frac{s^*}{\tau^*} & = & c, \\ -c^T \frac{x^*}{\tau^*} + b^T \frac{y^*}{\tau^*} & = & 0, \\ x^*/\tau^* & \in & \mathcal{K}, \\ s^*/\tau^* & \in & \mathcal{K}^*. \end{array}$$

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

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

is a primal-dual optimal solution.

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

$$\begin{array}{rcl} Ax^* & = & 0, \\ A^T y^* + s^* & = & 0, \\ -c^T x^* + b^T y^* & = & \kappa^*, \\ x^* & \in & \mathcal{K}, \\ s^* & \in & \mathcal{K}^*. \end{array}$$

This implies that at least one of

$$c^T x^* < 0 \tag{13.9}$$

or

$$b^T y^* > 0 \tag{13.10}$$

holds. If (13.9) is satisfied, then x^* is a certificate of dual infeasibility, whereas if (13.10) holds then y^* is a certificate of primal infeasibility.

In summary, by computing an appropriate solution to the homogeneous model, all information required for a solution to the original problem is obtained. A solution to the homogeneous model can be computed using a primal-dual interior-point algorithm *[And09]*.

13.3.2 Interior-point Termination Criterion

Since computations are performed in finite precision, and for efficiency reasons, it is not possible to solve the homogeneous model exactly in general. Hence, an exact optimal solution or an exact infeasibility certificate cannot be computed and a reasonable termination criterion has to be employed.

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

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

to the homogeneous model is generated, where

$$x^k \in \mathcal{K}, s^k \in \mathcal{K}^*, \tau^k, \kappa^k > 0.$$

Therefore, it is possible to compute the values:

$$\begin{array}{rcl} \rho_p^k & = & \arg\min_\rho \left\{\rho \mid \left\|A\frac{x^k}{\tau^k} - b\right\|_\infty \leq \rho\varepsilon_p(1 + \|b\|_\infty)\right\}, \\ \rho_d^k & = & \arg\min_\rho \left\{\rho \mid \left\|A^T\frac{y^k}{\tau^k} + \frac{s^k}{\tau^k} - c\right\|_\infty \leq \rho\varepsilon_d(1 + \|c\|_\infty)\right\}, \\ \rho_g^k & = & \arg\min_\rho \left\{\rho \mid \left(\frac{(x^k)^T s^k}{(\tau^k)^2}, \left|\frac{c^T x^k}{\tau^k} - \frac{b^T y^k}{\tau^k}\right|\right) \leq \rho\varepsilon_g \max\left(1, \frac{\min(|c^T x^k|, |b^T y^k|)}{\tau^k}\right)\right\}, \\ \rho_{pi}^k & = & \arg\min_\rho \left\{\rho \mid \left\|A^T y^k + s^k\right\|_\infty \leq \rho\varepsilon_i b^T y^k,\ b^T y^k > 0\right\} \text{ and} \\ \rho_{di}^k & = & \arg\min_\rho \left\{\rho \mid \left\|Ax^k\right\|_\infty \leq -\rho\varepsilon_i c^T x^k,\ c^T x^k < 0\right\}. \end{array}$$

Note $\varepsilon_p, \varepsilon_d, \varepsilon_g$ and ε_i are nonnegative user specified tolerances.

Optimal Case

Observe ρ_p^k measures how far x^k/τ^k is from being a good approximate primal feasible solution. Indeed if $\rho_p^k \leq 1$, then

$$\left\|A\frac{x^k}{\tau^k} - b\right\|_\infty \leq \varepsilon_p(1 + \|b\|_\infty). \tag{13.11}$$

This shows the violations in the primal equality constraints for the solution x^k/τ^k is small compared to the size of b given ε_p is small.

Similarly, if $\rho_d^k \leq 1$, then $(y^k, s^k)/\tau^k$ is an approximate dual feasible solution. If in addition $\rho_g \leq 1$, then the solution $(x^k, y^k, s^k)/\tau^k$ is approximate optimal because the associated primal and dual objective values are almost identical.

In other words if $\max(\rho_p^k, \rho_d^k, \rho_g^k) \leq 1$, then

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

is an approximate optimal solution.

Dual Infeasibility Certificate

Next assume that $\rho_{di}^k \leq 1$ and hence

$$\left\|Ax^k\right\|_\infty \leq -\varepsilon_i c^T x^k \text{ and } -c^T x^k > 0$$

holds. Now in this case the problem is declared dual infeasible and x^k is reported as a certificate of dual infeasibility. The motivation for this stopping criterion is as follows. Let

$$\bar{x} := \frac{x^k}{-c^T x^k}$$

and it is easy to verify that

$$\|A\bar{x}\|_\infty \leq \varepsilon_i \text{ and } c^T \bar{x} = -1$$

which shows $\bar{x}$ is an approximate certificate of dual infeasibility, where ε_i controls the quality of the approximation.

Primal Infeasiblity Certificate

Next assume that $\rho_{pi}^k \leq 1$ and hence

$$\left\|A^T y^k + s^k\right\|_\infty \leq \varepsilon_i b^T y^k \text{ and } b^T y^k > 0$$

holds. Now in this case the problem is declared primal infeasible and (y^k, s^k) is reported as a certificate of primal infeasibility. The motivation for this stopping criterion is as follows. Let

$$\bar{y} := \frac{y^k}{b^T y^k} \text{ and } \bar{s} := \frac{s^k}{b^T y^k}$$

and it is easy to verify that

$$\left\|A^T \bar{y} + \bar{s}\right\|_\infty \leq \varepsilon_i \text{ and } b^T \bar{y} = 1$$

which shows (y^k, s^k) is an approximate certificate of dual infeasibility, where ε_i controls the quality of the approximation.

13.3.3 Adjusting optimality criteria

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

Table 13.2: Parameters employed in termination criterion

ToleranceParameter	name
ε_p	*dparam.intpnt_co_tol_pfeas*
ε_d	*dparam.intpnt_co_tol_dfeas*
ε_g	*dparam.intpnt_co_tol_rel_gap*
ε_i	*dparam.intpnt_co_tol_infeas*

The default values of the termination tolerances are chosen such that for a majority of problems appearing in practice it is not possible to achieve much better accuracy. Therefore, tightening the tolerances usually is not worthwhile. However, an inspection of (13.11) reveals that the quality of the solution depends on $\|b\|_\infty$ and $\|c\|_\infty$; the smaller the norms are, the better the solution accuracy.

The interior-point method as implemented by **MOSEK** will converge toward optimality and primal and dual feasibility at the same rate *[And09]*. This means that if the optimizer is stopped prematurely then it is very unlikely that either the primal or dual solution is feasible. Another consequence is that in most cases all the tolerances, ε_p, ε_d, ε_g and ε_i, have to be relaxed together to achieve an effect.

If the optimizer terminates without locating a solution that satisfies the termination criteria, for example because of a stall or other numerical issues, then it will check if the solution found up to that point satisfies the same criteria with all tolerances multiplied by the value of *dparam.intpnt_co_tol_near_rel*. If this is the case, the solution is still declared as optimal.

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

13.3.4 The Interior-point Log

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

```
Optimizer  - threads                : 20
Optimizer  - solved problem         : the primal
Optimizer  - Constraints            : 1
Optimizer  - Cones                  : 2
Optimizer  - Scalar variables       : 6                 conic                  : 6
Optimizer  - Semi-definite variables: 0                 scalarized             : 0
Factor     - setup time             : 0.00              dense det. time        : 0.00
Factor     - ML order time          : 0.00              GP order time          : 0.00
Factor     - nonzeros before factor : 1                 after factor           : 1
Factor     - dense dim.             : 0                 flops                  : 1.70e+01
ITE PFEAS    DFEAS    GFEAS    PRSTATUS   POBJ              DOBJ              MU       TIME
0   1.0e+00  2.9e-01  3.4e+00  0.00e+00   2.414213562e+00   0.000000000e+00   1.0e+00  0.01
1   2.7e-01  7.9e-02  2.2e+00  8.83e-01   6.969257574e-01   -9.685901771e-03  2.7e-01  0.01
2   6.5e-02  1.9e-02  1.2e+00  1.16e+00   7.606090061e-01   6.046141322e-01   6.5e-02  0.01
3   1.7e-03  5.0e-04  2.2e-01  1.12e+00   7.084385672e-01   7.045122560e-01   1.7e-03  0.01
4   1.4e-08  4.2e-09  4.9e-08  1.00e+00   7.071067941e-01   7.071067599e-01   1.4e-08  0.01
```

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

Using the same notation as in Sec. 13.3.1 the columns of the iteration log have the following meaning:

- `ITE`: Iteration index k.
- `PFEAS`: $\left\|Ax^k - b\tau^k\right\|_\infty$. The numbers in this column should converge monotonically towards zero but may stall at low level due to rounding errors.
- `DFEAS`: $\left\|A^Ty^k + s^k - c\tau^k\right\|_\infty$. The numbers in this column should converge monotonically towards zero but may stall at low level due to rounding errors.
- `GFEAS`: $|-c^Tx^k + b^Ty^k - \kappa^k|$. The numbers in this column should converge monotonically towards zero but may stall at low level due to rounding errors.
- `PRSTATUS`: This number converges to 1 if the problem has an optimal solution whereas it converges to -1 if that is not the case.
- `POBJ`: c^Tx^k/τ^k. An estimate for the primal objective value.
- `DOBJ`: b^Ty^k/τ^k. An estimate for the dual objective value.
- `MU`: $\frac{(x^k)^Ts^k+\tau^k\kappa^k}{n+1}$. The numbers in this column should always converge to zero.
- `TIME`: Time spent since the optimization started (in seconds).

13.4 The Optimizer for Mixed-integer Problems

A problem is a mixed-integer optimization problem when one or more of the variables are constrained to be integer valued. Readers unfamiliar with integer optimization are recommended to consult some relevant literature, e.g. the book *[Wol98]* by Wolsey.

13.4.1 The Mixed-integer Optimizer Overview

MOSEK can solve mixed-integer

- linear,
- quadratic and quadratically constrained, and
- conic

problems, except for mixed-integer semidefinite problems. The mixed-integer optimizer is specialized for solving linear and conic optimization problems. Pure quadratic and quadratically constrained problems are automatically converted to conic form.

By default the mixed-integer optimizer is run-to-run deterministic. This means that if a problem is solved twice on the same computer with identical parameter settings and no time limit then the obtained solutions will be identical. If a time limit is set then this may not be case since the time taken to solve a problem is not deterministic. The mixed-integer optimizer is parallelized i.e. it can exploit multiple cores during the optimization.

The solution process can be split into these phases:

1. **Presolve:** See Sec. 13.1.
2. **Cut generation:** Valid inequalities (cuts) are added to improve the lower bound.
3. **Heuristic:** Using heuristics the optimizer tries to guess a good feasible solution. Heuristics can be controlled by the parameter *iparam.mio_heuristic_level* .
4. **Search:** The optimal solution is located by branching on integer variables.

13.4.2 Relaxations and bounds

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 (solving mixed-integer problems is NP-hard). For instance, a problem with n binary variables, may require time proportional to 2^n . The value of 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 on locating an optimal solution. Even if the problem is only solved approximately, it is important to know how far the approximate solution is from an optimal one. In order to say something about the quality of an approximate solution the concept of *relaxation* is important.

Consider for example a mixed-integer optimization problem

$$\begin{array}{rccccc} z^* & = & \text{minimize} & c^T x & & \\ & & \text{subject to} & Ax & = & b, \\ & & & x \geq 0 & & \\ & & & x_j \in \mathbb{Z}, & & \forall j \in \mathcal{J}. \end{array} \tag{13.12}$$

It has the continuous relaxation

$$\begin{array}{rccccc} \underline{z} & = & \text{minimize} & c^T x & & \\ & & \text{subject to} & Ax & = & b, \\ & & & x \geq 0 & & \end{array} \tag{13.13}$$

obtained simply by ignoring the integrality restrictions. The relaxation is a continuous problem, and therefore much faster to solve to optimality with a linear (or, in the general case, conic) optimizer. We call the optimal value $\underline{z}$ the *objective bound*. The objective bound $\underline{z}$ normally increases during the solution search process when the continuous relaxation is gradually refined.

Moreover, if $\hat{x}$ is any feasible solution to (13.12) and

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

then

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

These two inequalities allow us to estimate the quality of the integer solution: it is no further away from the optimum than $\bar{z} - \underline{z}$ in terms of the objective value. Whenever a mixed-integer problem is solved **MOSEK** reports this lower bound so that the quality of the reported solution can be evaluated.

13.4.3 Outer approximation for mixed-integer conic problems

The relaxations of mixed integer conic problems can be solved either as a nonlinear problem with the interior point algorithm (default) or with a linear outer approximation algorithm. The type of relaxation used can be set with *iparam.mio_conic_outer_approximation*. The best value for this option is highly problem dependent.

13.4.4 Randomization

A number of internal algorithms of the mixed-integer solver are dependend on random tie-breaking. The random tie-breaking can have a significant impact on the path taken by the algorithm and the optimal solution returned. The random seed can be set with the parameter *iparam.mio_seed*.

13.4.5 Termination Criterion

In general, it is time consuming to find an exact feasible and optimal solution to an integer optimization problem, though in many practical cases it may be possible to find a sufficiently good solution. The issue of terminating the mixed-integer optimizer is rather delicate and the user has numerous possibilities of influencing it with various parameters. The mixed-integer optimizer employs a relaxed feasibility and optimality criterion to determine when a satisfactory solution is located.

A candidate solution that is feasible for the continuous relaxation is said to be an *integer feasible solution* if the criterion

$$\min(x_j - \lfloor x_j \rfloor, \lceil x_j \rceil - x_j) \leq \delta_1 \quad \forall j \in \mathcal{J}$$

is satisfied, meaning that x_j is at most δ_1 from the nearest integer.

Whenever the integer optimizer locates an integer feasible solution it will check if the criterion

$$\bar{z} - \underline{z} \leq \max(\delta_2, \delta_3 \max(\delta_4, |\bar{z}|))$$

is satisfied. If this is the case, the integer optimizer terminates and reports the integer feasible solution as an optimal solution.

All the δ tolerances discussed above can be adjusted using suitable parameters — see Table 13.3.

Table 13.3: Tolerances for the mixed-integer optimizer.

Tolerance	Parameter name
δ_1	*dparam.mio_tol_abs_relax_int*
δ_2	*dparam.mio_tol_abs_gap*
δ_3	*dparam.mio_tol_rel_gap*
δ_4	*dparam.mio_rel_gap_const*

In Table 13.4 some other common parameters affecting the integer optimizer termination criterion are shown.

Table 13.4: Other parameters affecting the integer optimizer termination criterion.

Parameter name	Explanation
iparam.mio_max_num_branches	Maximum number of branches allowed.
iparam.mio_max_num_relaxs	Maximum number of relaxations allowed.
iparam.mio_max_num_solutions	Maximum number of feasible integer solutions allowed.

13.4.6 Speeding 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 Sec. 13.4.5 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. See Sec. 6.7.2.
- 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 *[Wol98]*.

13.4.7 Understanding Solution Quality

To determine the quality of the solution one should check the following:

- The problem status and solution status returned by **MOSEK**, as well as constraint violations in case of suboptimal solutions.
- The *optimality gap* defined as

$$\epsilon = |(\text{objective value of feasible solution}) - (\text{objective bound})| = |\bar{z} - \underline{z}|.$$

  which measures how much the located solution can deviate from the optimal solution to the problem. The optimality gap can be retrieved through the information item *dinfitem.mio_obj_abs_gap*. Often it is more meaningful to look at the relative optimality gap normalized against the magnitude of the solution.

$$\epsilon_{\text{rel}} = \frac{|\bar{z} - \underline{z}|}{\max(\delta_4, |\bar{z}|)}.$$

  The relative optimality gap is available in the information item *dinfitem.mio_obj_rel_gap*.

13.4.8 The Mixed-integer Log

Below is a typical log output from the mixed-integer optimizer:

```
Presolved problem: 6573 variables, 35728 constraints, 101258 non-zeros
Presolved problem: 0 general integer, 4294 binary, 2279 continuous
Clique table size: 1636
BRANCHES RELAXS   ACT_NDS  DEPTH    BEST_INT_OBJ         BEST_RELAX_OBJ       REL_GAP(%)  TIME
0        1        0        0        NA                   1.8218819866e+07     NA          1.6
0        1        0        0        1.8331557950e+07     1.8218819866e+07     0.61        3.5
0        1        0        0        1.8300507546e+07     1.8218819866e+07     0.45        4.3
Cut generation started.
0        2        0        0        1.8300507546e+07     1.8218819866e+07     0.45        5.3
Cut generation terminated. Time = 1.43
0        3        0        0        1.8286893047e+07     1.8231580587e+07     0.30        7.5
15       18       1        0        1.8286893047e+07     1.8231580587e+07     0.30        10.5
31       34       1        0        1.8286893047e+07     1.8231580587e+07     0.30        11.1
51       54       1        0        1.8286893047e+07     1.8231580587e+07     0.30        11.6
91       94       1        0        1.8286893047e+07     1.8231580587e+07     0.30        12.4
171      174      1        0        1.8286893047e+07     1.8231580587e+07     0.30        14.3
331      334      1        0        1.8286893047e+07     1.8231580587e+07     0.30        17.9

[ ... ]

Objective of best integer solution : 1.825846762609e+07
Best objective bound               : 1.823311032986e+07
Construct solution objective       : Not employed
Construct solution # roundings     : 0
User objective cut value           : 0
Number of cuts generated           : 117
  Number of Gomory cuts            : 108
  Number of CMIR cuts              : 9
```

(continues on next page)

(continued from previous page)

```
Number of branches               : 4425
Number of relaxations solved     : 4410
Number of interior point iterations: 25
Number of simplex iterations     : 221131
```

The first lines contain a summary of the problem as seen by the optimizer. This is followed by the iteration log. The columns have the following meaning:

- `BRANCHES`: Number of branches generated.
- `RELAXS`: Number of relaxations solved.
- `ACT_NDS`: Number of active branch bound nodes.
- `DEPTH`: Depth of the recently solved node.
- `BEST_INT_OBJ`: The best integer objective value, $\bar{z}$.
- `BEST_RELAX_OBJ`: The best objective bound, $\underline{z}$.
- `REL_GAP(%)`: Relative optimality gap, $100\% \cdot \epsilon_{\mathrm{rel}}$
- `TIME`: Time (in seconds) from the start of optimization.

Following that a summary of the optimization process is printed.

Chapter 14

Additional features

In this section we describe additional features and tools which enable more detailed analysis of optimization problems with **MOSEK**.

14.1 Problem Analyzer

The problem analyzer prints a survey of the structure of the problem, with information about linear constraints and objective, quadratic constraints, conic constraints and variables.

In the initial stages of model formulation the problem analyzer may be used as a quick way of verifying that the model has been built or imported correctly. In later stages it can help revealing special structures within the model that may be used to tune the optimizer's performance or to identify the causes of numerical difficulties.

The problem analyzer is run using *Task.analyzeproblem*. It prints its output to a log stream. The output is similar to the one below (this is the problem survey of the aflow30a problem from the MIPLIB 2003 collection).

```
                    Analyzing the problem

*** Structural report
               Dimensions
               Constraints    Variables     Matrix var.    Cones
               479            842           0              0

               Constraint and bound types
               Free           Lower         Upper          Ranged         Fixed
 Constraints:  0              0             421            0              58
   Variables:  0              0             0              842            0

               Integer constraint types
               Binary         General
               421            0

*** Data report
               Nonzeros       Min           Max
        |cj|:  421            1.1e+01       5.0e+02
       |Aij|:  2091           1.0e+00       1.0e+02

               # finite       Min           Max
      |blci|:  58             1.0e+00       1.0e+01
      |buci|:  479            0.0e+00       1.0e+01
      |blxj|:  842            0.0e+00       0.0e+00
      |buxj|:  842            1.0e+00       1.0e+02

*** Done analyzing the problem
```

The survey is divided into a structural and numerical report. The content should be self-explanatory.

14.2 Automatic Repair of Infeasible Problems

MOSEK provides an automatic repair tool for infeasible linear problems which we cover in this section. Note that most infeasible models are so due to bugs which can (and should) be more reliably fixed manually, using the knowledge of the model structure. We discuss this approach in Sec. 8.3.

14.2.1 Automatic repair

The main idea can be described as follows. Consider the linear optimization problem with m constraints and n variables

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

which is assumed to be infeasible.

One way of making the problem feasible is to reduce the lower bounds and increase the upper bounds. If the change is sufficiently large the problem becomes feasible. Now an obvious idea is to compute the optimal relaxation by solving an optimization problem. The problem

$$\begin{array}{lccccc}
\text{minimize} & & & p(v_l^c, v_u^c, v_l^x, v_u^x) & & \\
\text{subject to} & l^c & \leq & Ax + v_l^c - v_u^c & \leq & u^c, \\
 & l^x & \leq & x + v_l^x - v_u^x & \leq & u^x, \\
 & & & v_l^c, v_u^c, v_l^x, v_u^x \geq 0 & &
\end{array} \tag{14.1}$$

does exactly that. The additional variables $(v_l^c)_i$, $(v_u^c)_i$, $(v_l^x)_j$ and $(v_u^c)_j$ are *elasticity* variables because they allow a constraint to be violated and hence add some elasticity to the problem. For instance, the elasticity variable $(v_l^c)_i$ controls how much the lower bound $(l^c)_i$ should be relaxed to make the problem feasible. Finally, the so-called penalty function

$$p(v_l^c, v_u^c, v_l^x, v_u^x)$$

is chosen so it penalizes changes to bounds. Given the weights

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

a natural choice is

$$p(v_l^c, v_u^c, v_l^x, v_u^x) = (w_l^c)^T v_l^c + (w_u^c)^T v_u^c + (w_l^x)^T v_l^x + (w_u^x)^T v_u^x.$$

Hence, the penalty function $p()$ is a weighted sum of the elasticity variables and therefore the problem (14.1) keeps the amount of relaxation at a minimum. Please observe that

- the problem (14.1) is always feasible.
- a negative weight implies problem (14.1) is unbounded. For this reason if the value of a weight is negative **MOSEK** fixes the associated elasticity variable to zero. Clearly, if one or more of the weights are negative, it may imply that it is not possible to repair the problem.

A simple choice of weights is to set them all to 1, but of course that does not take into account that constraints may have different importance.

Caveats

Observe if the infeasible problem

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

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

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

Using the automatic repair tool

In this subsection we consider an infeasible linear optimization example:

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

The function *Task.primalrepair* can be used to repair an infeasible problem. This can be used for linear and conic optimization problems, possibly with integer variables.

Listing 14.1: An example of feasibility repair applied to problem (14.2).

```
import sys
import mosek

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

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


def main(inputfile):
    # Make a MOSEK environment
    with mosek.Env() as env:
        with env.Task(0, 0) as task:
            # Attach a printer to the task
            task.set_Stream(mosek.streamtype.log, streamprinter)

            # Read data
            task.readdata(inputfile)

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

            task.primalrepair(None, None, None, None)

            sum_viol = task.getdouinf(mosek.dinfitem.primal_repair_penalty_obj)
            print("Minimized sum of violations = %e" % sum_viol)

            task.optimize()
```

(continues on next page)

(continued from previous page)

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

# call the main function
try:
    filename = "../data/feasrepair.lp"
    if len(sys.argv) > 1:
        filename = sys.argv[1]
    main(filename)
except Exception as e:
    print(e)
    raise
```

The above code will produce the following log report:

```
MOSEK Version 9.0.0.25(ALPHA) (Build date: 2017-11-7 16:11:50)
Copyright (c) MOSEK ApS, Denmark. WWW: mosek.com
Platform: Linux/64-X86

Open file 'feasrepair.lp'
Reading started.
Reading terminated. Time: 0.00

Read summary
  Type             : LO (linear optimization problem)
  Objective sense  : min
  Scalar variables : 2
  Matrix variables : 0
  Constraints      : 4
  Cones            : 0
  Time             : 0.0

Problem
  Name                   :
  Objective sense        : min
  Type                   : LO (linear optimization problem)
  Constraints            : 4
  Cones                  : 0
  Scalar variables       : 2
  Matrix variables       : 0
  Integer variables      : 0

Primal feasibility repair started.
Optimizer started.
Presolve started.
Linear dependency checker started.
Linear dependency checker terminated.
Eliminator started.
Freed constraints in eliminator : 2
Eliminator terminated.
Eliminator - tries                  : 1                 time                   : 0.00
Lin. dep.  - tries                  : 1                 time                   : 0.00
Lin. dep.  - number                 : 0
Presolve terminated. Time: 0.00
Problem
  Name                   :
  Objective sense        : min
  Type                   : LO (linear optimization problem)
  Constraints            : 8
  Cones                  : 0
  Scalar variables       : 14
  Matrix variables       : 0
```

(continues on next page)

(continued from previous page)

```
  Integer variables      : 0

Optimizer  - threads                : 20
Optimizer  - solved problem         : the primal
Optimizer  - Constraints            : 2
Optimizer  - Cones                  : 0
Optimizer  - Scalar variables       : 5                 conic                  : 0
Optimizer  - Semi-definite variables: 0                 scalarized             : 0
Factor     - setup time             : 0.00              dense det. time        : 0.00
Factor     - ML order time          : 0.00              GP order time          : 0.00
Factor     - nonzeros before factor : 3                 after factor           : 3
Factor     - dense dim.             : 0                 flops                  : 5.00e+01
ITE PFEAS    DFEAS    GFEAS    PRSTATUS   POBJ              DOBJ              MU       TIME
0   2.7e+01  1.0e+00  4.0e+00  1.00e+00   3.000000000e+00   0.000000000e+00   1.0e+00  0.00
1   2.5e+01  9.1e-01  1.4e+00  0.00e+00   8.711262850e+00   1.115287830e+01   2.4e+00  0.00
2   2.4e+00  8.8e-02  1.4e-01  -7.33e-01  4.062505701e+01   4.422203730e+01   2.3e-01  0.00
3   9.4e-02  3.4e-03  5.5e-03  1.33e+00   4.250700434e+01   4.258548510e+01   9.1e-03  0.00
4   2.0e-05  7.2e-07  1.1e-06  1.02e+00   4.249996599e+01   4.249998669e+01   1.9e-06  0.00
5   2.0e-09  7.2e-11  1.1e-10  1.00e+00   4.250000000e+01   4.250000000e+01   1.9e-10  0.00
Basis identification started.
Basis identification terminated. Time: 0.00
Optimizer terminated. Time: 0.01

Basic solution summary
  Problem status  : PRIMAL_AND_DUAL_FEASIBLE
  Solution status : OPTIMAL
  Primal.  obj: 4.2500000000e+01    nrm: 6e+02    Viol.  con: 1e-13    var: 0e+00
  Dual.    obj: 4.2499999999e+01    nrm: 2e+00    Viol.  con: 0e+00    var: 9e-11
Optimal objective value of the penalty problem: 4.250000000000e+01

Repairing bounds.
Increasing the upper bound 1.35e+02 on constraint 'c4' (3) with 2.25e+01.
Decreasing the lower bound 6.50e+02 on variable 'x2' (4) with 2.00e+01.
Primal feasibility repair terminated.
Optimizer started.
Optimizer terminated. Time: 0.00


Interior-point solution summary
  Problem status  : PRIMAL_AND_DUAL_FEASIBLE
  Solution status : OPTIMAL
  Primal.  obj: -5.6700000000e+03   nrm: 6e+02    Viol.  con: 0e+00    var: 0e+00
  Dual.    obj: -5.6700000000e+03   nrm: 1e+01    Viol.  con: 0e+00    var: 0e+00

Basic solution summary
  Problem status  : PRIMAL_AND_DUAL_FEASIBLE
  Solution status : OPTIMAL
  Primal.  obj: -5.6700000000e+03   nrm: 6e+02    Viol.  con: 0e+00    var: 0e+00
  Dual.    obj: -5.6700000000e+03   nrm: 1e+01    Viol.  con: 0e+00    var: 0e+00

Optimizer summary
  Optimizer                 -                        time: 0.00
    Interior-point          - iterations : 0         time: 0.00
      Basis identification  -                        time: 0.00
        Primal              - iterations : 0         time: 0.00
        Dual                - iterations : 0         time: 0.00
        Clean primal        - iterations : 0         time: 0.00
        Clean dual          - iterations : 0         time: 0.00
    Simplex                 -                        time: 0.00
      Primal simplex        - iterations : 0         time: 0.00
      Dual simplex          - iterations : 0         time: 0.00
```

(continues on next page)

(continued from previous page)

```
Mixed integer          - relaxations: 0        time: 0.00
```

It will also modify the task according to the optimal elasticity variables found. In this case the optimal repair it is to increase the upper bound on constraint `c4` by 22.5 and decrease the lower bound on variable `x2` by 20.

14.3 Sensitivity Analysis

Given an optimization problem it is often useful to obtain information about how the optimal objective value changes when the problem parameters are perturbed. E.g, assume that a bound represents the capacity of a machine. Now, it may be possible to expand the capacity for a certain cost and hence it is worthwhile knowing what the value of additional capacity is. This is precisely the type of questions the sensitivity analysis deals with.

Analyzing how the optimal objective value changes when the problem data is changed is called *sensitivity analysis*.

References

The book *[Chv83]* discusses the classical sensitivity analysis in Chapter 10 whereas the book *[RTV97]* presents a modern introduction to sensitivity analysis. Finally, it is recommended to read the short paper *[Wal00]* to avoid some of the pitfalls associated with sensitivity analysis.

> **Warning:** Currently, sensitivity analysis is only available for continuous linear optimization problems. Moreover, **MOSEK** can only deal with perturbations of bounds and objective function coefficients.

14.3.1 Sensitivity Analysis for Linear Problems

The Optimal Objective Value Function

Assume that we are given the problem

$$\begin{array}{rclccccl} z(l^c,u^c,l^x,u^x,c) & = & \text{minimize} & & & c^T x & & \\ & & \text{subject to} & l^c & \leq & Ax & \leq & u^c, \\ & & & l^x & \leq & x & \leq & u^x, \end{array} \tag{14.3}$$

and we want to know how the optimal objective value changes as l_i^c is perturbed. To answer this question we define the perturbed problem for l_i^c as follows

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

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

$$f_{l_i^c}(\beta) \tag{14.4}$$

shows the optimal objective value as a function of β. Please note that a change in β corresponds to a perturbation in l_i^c and hence (14.4) shows the optimal objective value as a function of varying l_i^c with the other bounds fixed.

It is possible to prove that the function (14.4) is a piecewise linear and convex function, i.e. its graph may look like in Fig. 14.1 and Fig. 14.2.

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

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

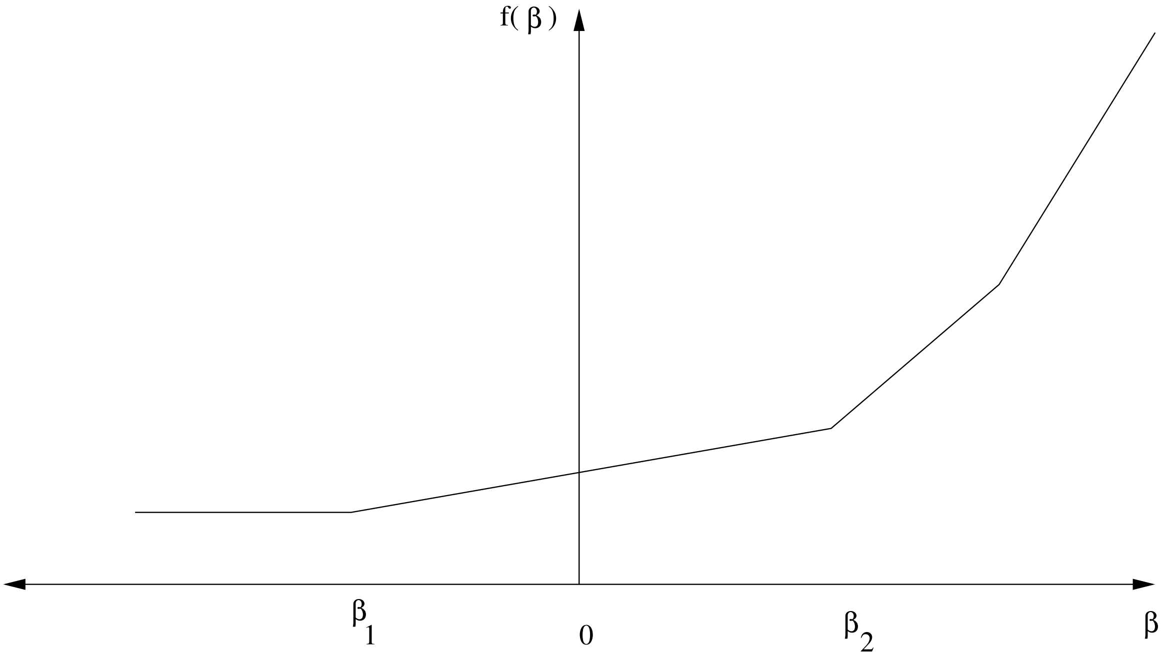


Fig. 14.1: $\beta = 0$ is in the interior of linearity interval.

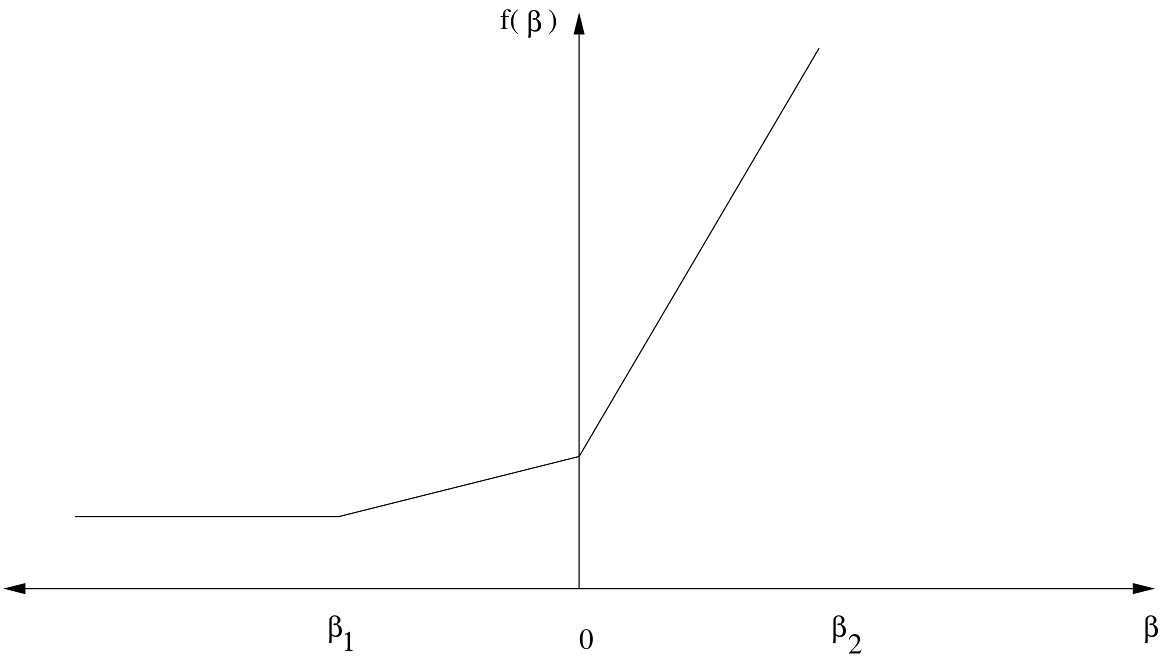


Fig. 14.2: $\beta = 0$ is a breakpoint.

which is called the *shadow price* related to l_i^c. The shadow price specifies how the objective value changes for small changes of β around zero. Moreover, we are interested in the *linearity interval*

$$\beta \in [\beta_1, \beta_2]$$

for which

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

Since $f_{l_i^c}$ is not a smooth function $f'_{l_i^c}$ may not be defined at 0, as illustrated in Fig. 14.2. In this case we can define a left and a right shadow price and a left and a right linearity interval.

The function $f_{l_i^c}$ considered only changes in l_i^c. We can define similar functions for the remaining parameters of the z defined in (14.3) as well:

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

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

Equality Constraints

In **MOSEK** a constraint can be specified as either an equality constraint or a ranged constraint. If some constraint e_i^c is an equality constraint, we define the optimal value function for this constraint as

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

Thus for an equality constraint the upper and the lower bounds (which are equal) are perturbed simultaneously. Therefore, **MOSEK** will handle sensitivity analysis differently for a ranged constraint with $l_i^c = u_i^c$ and for an equality constraint.

The Basis Type Sensitivity Analysis

The classical sensitivity analysis discussed in most textbooks about linear optimization, e.g. *[Chv83]*, is based on an optimal basis. This method may produce misleading results *[RTV97]* but is computationally cheap. This is the type of sensitivity analysis implemented in **MOSEK**.

We will now briefly discuss the basis type sensitivity analysis. Given an optimal basic solution which provides a partition of variables into basic and non-basic variables, the basis type sensitivity analysis computes the linearity interval $[\beta_1, \beta_2]$ so that the basis remains optimal for the perturbed problem. A shadow price associated with the linearity interval is also computed. However, it is well-known that an optimal basic solution may not be unique and therefore the result depends on the optimal basic solution employed in the sensitivity analysis. If the optimal objective value function has a breakpoint for $\beta = 0$ then the basis type sensitivity method will only provide a subset of either the left or the right linearity interval.

In summary, the basis type sensitivity analysis is computationally cheap but does not provide complete information. Hence, the results of the basis type sensitivity analysis should be used with care.

Example: Sensitivity Analysis

As an example we will use the following transportation problem. Consider the problem of minimizing the transportation cost between a number of production plants and stores. Each plant supplies a number of goods and each store has a given demand that must be met. Supply, demand and cost of transportation per unit are shown in Fig. 14.3.

If we denote the number of transported goods from location i to location j by x_{ij}, problem can be formulated as the linear optimization problem of minimizing

$$1x_{11} \quad + \quad 2x_{12} \quad + \quad 5x_{23} \quad + \quad 2x_{24} \quad + \quad 1x_{31} \quad + \quad 2x_{33} \quad + \quad 1x_{34}$$

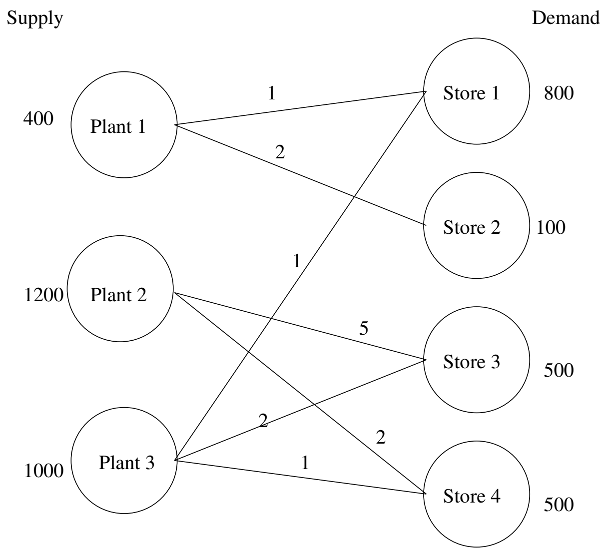


Fig. 14.3: Supply, demand and cost of transportation.

subject to

$$
\begin{array}{ccccccccccccccc}
x_{11} & + & x_{12} & & & & & & & & & & & \leq & 400, \\
& & & & x_{23} & + & x_{24} & & & & & & & \leq & 1200, \\
& & & & & & & & x_{31} & + & x_{33} & + & x_{34} & \leq & 1000, \\
x_{11} & & & & & & & + & x_{31} & & & & & = & 800, \\
& & x_{12} & & & & & & & & & & & = & 100, \\
& & & & x_{23} & + & & & & & x_{33} & & & = & 500, \\
& & & & & & x_{24} & + & & & & & x_{34} & = & 500, \\
x_{11}, & & x_{12}, & & x_{23}, & & x_{24}, & & x_{31}, & & x_{33}, & & x_{34} & \geq & 0.
\end{array}
\tag{14.5}
$$

The sensitivity parameters are shown in Table 14.1 and Table 14.2.

Table 14.1: Ranges and shadow prices related to bounds on constraints and variables.

Con.	β_1	β_2	σ_1	σ_2
1	−300.00	0.00	3.00	3.00
2	−700.00	$+\infty$	0.00	0.00
3	−500.00	0.00	3.00	3.00
4	−0.00	500.00	4.00	4.00
5	−0.00	300.00	5.00	5.00
6	−0.00	700.00	5.00	5.00
7	−500.00	700.00	2.00	2.00
Var.	β_1	β_2	σ_1	σ_2
x_{11}	$-\infty$	300.00	0.00	0.00
x_{12}	$-\infty$	100.00	0.00	0.00
x_{23}	$-\infty$	0.00	0.00	0.00
x_{24}	$-\infty$	500.00	0.00	0.00
x_{31}	$-\infty$	500.00	0.00	0.00
x_{33}	$-\infty$	500.00	0.00	0.00
x_{34}	−0.000000	500.00	2.00	2.00

Table 14.2: Ranges and shadow prices related to the objective coefficients.

Var.	β_1	β_2	σ_1	σ_2
c_1	$-\infty$	3.00	300.00	300.00
c_2	$-\infty$	∞	100.00	100.00
c_3	-2.00	∞	0.00	0.00
c_4	$-\infty$	2.00	500.00	500.00
c_5	-3.00	∞	500.00	500.00
c_6	$-\infty$	2.00	500.00	500.00
c_7	-2.00	∞	0.00	0.00

Examining the results from the sensitivity analysis we see that for constraint number 1 we have $\sigma_1 = 3$ and $\beta_1 = -300,\ \beta_2 = 0$.

If the upper bound on constraint 1 is decreased by

$$\beta \in [0, 300]$$

then the optimal objective value will increase by the value

$$\sigma_1 \beta = 3\beta.$$

14.3.2 Sensitivity Analysis with MOSEK

MOSEK provides the functions *Task.primalsensitivity* and *Task.dualsensitivity* for performing sensitivity analysis. The code in Listing 14.2 gives an example of its use.

Listing 14.2: Example of sensitivity analysis with the **MOSEK** Optimizer API for Python.

```
import sys
import mosek

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

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


def main():
    # Create a MOSEK environment
    with mosek.Env() as env:
        # Attach a printer to the environment
        env.set_Stream(mosek.streamtype.log, streamprinter)

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

            # Set up data
            bkc = [mosek.boundkey.up, mosek.boundkey.up,
                   mosek.boundkey.up, mosek.boundkey.fx,
                   mosek.boundkey.fx, mosek.boundkey.fx,
```

(continues on next page)

(continued from previous page)

```
        mosek.boundkey.fx]
blc = [-inf, -inf, -inf, 800., 100., 500., 500.]
buc = [400., 1200., 1000., 800., 100., 500., 500.]

bkx = [mosek.boundkey.lo, mosek.boundkey.lo,
       mosek.boundkey.lo, mosek.boundkey.lo,
       mosek.boundkey.lo, mosek.boundkey.lo,
       mosek.boundkey.lo]
c = [1.0, 2.0, 5.0, 2.0, 1.0, 2.0, 1.0]
blx = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
bux = [inf, inf, inf, inf, inf, inf, inf]

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

val = [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
       1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]

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

# Input linear data
task.inputdata(numcon, numvar,
               c, 0.0,
               ptrb, ptre, sub, val,
               bkc, blc, buc,
               bkx, blx, bux)
# Set objective sense
task.putobjsense(mosek.objsense.minimize)

# Optimize
task.optimize()

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

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

leftpricei = [0., 0.]
rightpricei = [0., 0.]
leftrangei = [0., 0.]
rightrangei = [0., 0.]
leftpricej = [0., 0.]
rightpricej = [0., 0.]
leftrangej = [0., 0.]
rightrangej = [0., 0.]

task.primalsensitivity(subi,
                       marki,
                       subj,
                       markj,
                       leftpricei,
                       rightpricei,
                       leftrangei,
                       rightrangei,
                       leftpricej,
```

(continues on next page)

(continued from previous page)

```
                                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 = [0., 0.]
            rightprice = [0., 0.]
            leftrange = [0., 0.]
            rightrange = [0., 0.]
            subc = [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.MosekException as e:
    print("ERROR: %s" % str(e.errno))
    if e.msg is not None:
        print("\t%s" % e.msg)
    sys.exit(1)
except:
    import traceback
    traceback.print_exc()
    sys.exit(1)
```

Chapter 15

API Reference

This section contains the complete reference of the **MOSEK** Optimizer API for Python. It is organized as follows:

- *General API conventions.*
- **Methods:**
  - *Class Env* (The **MOSEK** environment)
  - *Class Task* (An optimization task)
  - *Browse by topic*
- **Optimizer parameters:**
  - *Double*, *Integer*, *String*
  - *Full list*
  - *Browse by topic*
- **Optimizer information items:**
  - *`Double`*, *`Integer`*, *`Long`*
- *Optimizer response codes*
- *Enumerations*
- *Exceptions*
- *User-defined function types*
- *Nonlinear API (SCopt)*

15.1 API Conventions

15.1.1 Function arguments

Naming Convention

In the definition of the **MOSEK** Optimizer API for Python a consistent naming convention has been used. This implies that whenever for example `numcon` is an argument in a function definition it indicates the number of constraints. In Table 15.1 the variable names used to specify the problem parameters are listed.

Table 15.1: Naming conventions used in the **MOSEK** Optimizer API for Python.

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

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

- The quadratic terms in the objective: $q^o_{\mathtt{qosubi[t]},\mathtt{qosubj[t]}} = \mathtt{qoval[t]}, \quad t = 0, \ldots, \mathtt{numqonz} - 1.$
- The linear terms in the objective : $c_j = \mathtt{c[j]}, \quad j = 0, \ldots, \mathtt{numvar} - 1$
- The fixed term in the objective : $c^f = \mathtt{cfix}$.
- The quadratic terms in the constraints: $q^{\mathtt{qcsubk[t]}}_{\mathtt{qcsubi[t]},\mathtt{qcsubj[t]}} = \mathtt{qcval[t]}, \quad t = 0, \ldots, \mathtt{numqcnz} - 1$
- The linear terms in the constraints: $a_{\mathtt{asub[t]},\mathtt{j}} = \mathtt{aval[t]}, \quad t = \mathtt{ptrb[j]}, \ldots, \mathtt{ptre[j]} - 1, \; j = 0, \ldots, \mathtt{numvar} - 1$

Information about input/output arguments

The following are purely informational tags which indicate how **MOSEK** treats a specific function argument.

- (input) An input argument. It is used to input data to **MOSEK**.
- (output) An output argument. It can be a user-preallocated data structure, a reference, a string buffer etc. where **MOSEK** will output some data.
- (input/output) An input/output argument. **MOSEK** will read the data and overwrite it with new/updated information.

15.1.2 Bounds

The bounds on the constraints and variables are specified using the variables `bkc`, `blc`, and `buc`. The components of the integer array `bkc` specify the bound type according to Table 15.2

Table 15.2: Symbolic key for variable and constraint bounds.

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

For instance `bkc[2]=`*boundkey.lo* means that $-\infty < l_2^c$ and $u_2^c = \infty$. Even if a variable or constraint is bounded only from below, e.g. $x \geq 0$, both bounds are inputted or extracted; the irrelevant value is ignored.

Finally, the numerical values of the bounds are given by

$$l_k^c = \texttt{blc}[\mathrm{k}], \quad k = 0, \ldots, \texttt{numcon} - 1$$

$$u_k^c = \texttt{buc}[\mathrm{k}], \quad k = 0, \ldots, \texttt{numcon} - 1.$$

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

$$l_j^x = \texttt{blx}[\mathrm{j}], \quad j = 0, \ldots, \texttt{numvar} - 1.$$

$$u_j^x = \texttt{bux}[\mathrm{j}], \quad j = 0, \ldots, \texttt{numvar} - 1.$$

15.1.3 Vector Formats

Three different vector formats are used in the **MOSEK** API:

Full (dense) vector

This is simply an array where the first element corresponds to the first item, the second element to the second item etc. For example to get the linear coefficients of the objective in `task` with `numvar` variables, one would write

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

Vector slice

A vector slice is a range of values from `first` up to and **not including** `last` entry in the vector, i.e. for the set of indices `i` such that `first <= i < last`. For example, to get the bounds associated with constrains 2 through 9 (both inclusive) one would write

```
upper_bound = zeros(8,float)
lower_bound = zeros(8,float)
bound_key   = array([0] * 8)

task.getconboundslice(2, 10,
                      bound_key,lower_bound,upper_bound)
```

Sparse vector

A sparse vector is given as an array of indexes and an array of values. The indexes need not be ordered. For example, to input a set of bounds associated with constraints number 1, 6, 3, and 9, one might write

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

15.1.4 Matrix Formats

The coefficient matrices in a problem are inputted and extracted in a sparse format. That means only the nonzero entries are listed.

Unordered Triplets

In unordered triplet format each entry is defined as a row index, a column index and a coefficient. For example, to input the A matrix coefficients for $a_{1,2} = 1.1, a_{3,3} = 4.3$, and $a_{5,4} = 0.2$, one would write as follows:

```
subi = 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 are modified — all other are unchanged. In other cases (such as *Task.putqconk*) the triplet format is used to modify *all* entries — entries that are not specified are set to 0.

Column or Row Ordered Sparse Matrix

In a sparse matrix format only the non-zero entries of the matrix are stored. **MOSEK** uses a sparse packed matrix format ordered either by columns or rows. Here we describe the column-wise format. The row-wise format is based on the same principle.

Column ordered sparse format

A sparse matrix in column ordered format is essentially a list of all non-zero entries read column by column from left to right and from top to bottom within each column. The exact representation uses four arrays:

- `asub`: Array of size equal to the number of nonzeros. List of row indexes.
- `aval`: Array of size equal to the number of nonzeros. List of non-zero entries of A ordered by columns.
- `ptrb`: Array of size `numcol`, where `ptrb[j]` is the position of the first value/index in `aval`/ `asub` for the j-th column.
- `ptre`: Array of size `numcol`, where `ptre[j]` is the position of the last value/index plus one in `aval` / `asub` for the j-th column.

With this representation the values of a matrix A with `numcol` columns are assigned using:

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

As an example consider the matrix

$$A = \begin{bmatrix} 1.1 & & 1.3 & 1.4 & \\ & 2.2 & & & 2.5 \\ 3.1 & & & 3.4 & \\ & & 4.4 & & \end{bmatrix} \tag{15.1}$$

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

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

Fig. 15.1 illustrates how the matrix A in (15.1) is represented in column ordered sparse matrix format.

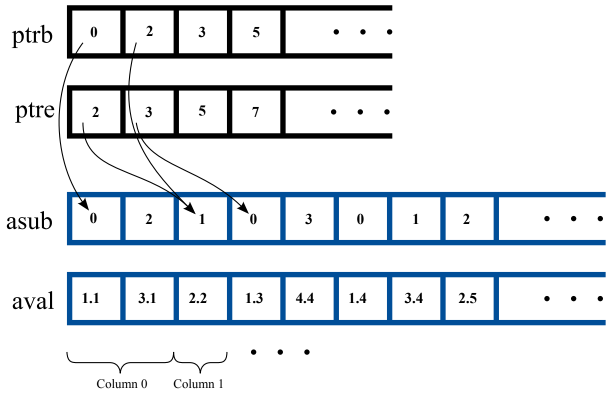


Fig. 15.1: The matrix A (15.1) represented in column ordered packed sparse matrix format.

Column ordered sparse format with nonzeros

Note that `nzc[j] := ptre[j]-ptrb[j]` is exactly the number of nonzero elements in the j-th column of A. In some functions a sparse matrix will be represented using the equivalent dataset `asub`, `aval`, `ptrb`, `nzc`. The matrix A (15.1) would now be represented as:

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

Row ordered sparse matrix

The matrix A (15.1) can also be represented in the row ordered sparse matrix format as:

$$
\begin{aligned}
\texttt{ptrb} &= [0, 3, 5, 7], \\
\texttt{ptre} &= [3, 5, 7, 8], \\
\texttt{asub} &= [0, 2, 3, 1, 4, 0, 3, 2], \\
\texttt{aval} &= [1.1, 1.3, 1.4, 2.2, 2.5, 3.1, 3.4, 4.4].
\end{aligned}
$$

15.2 Functions grouped by topic

Callback

- *Task.set_InfoCallback* – Receive callbacks with solver status and information during optimization.
- *Task.set_Progress* – Receive callbacks about current status of the solver during optimization.
- *Task.set_Stream* – Directs all output from a task stream to a callback function.
- *Infrequent: Env.set_Stream*

Environment and task management

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

Infeasibility diagnostic

- *Task.getinfeasiblesubproblem* – Obtains an infeasible subproblem.
- *Task.primalrepair* – Repairs a primal infeasible optimization problem by adjusting the bounds on the constraints and variables.

Information items and statistics

- *Task.getdouinf* – Obtains a double information item.
- *Task.getintinf* – Obtains an integer information item.
- *Task.getlintinf* – Obtains a long integer information item.
- *Task.updatesolutioninfo* – Update the information items related to the solution.

Input/Output

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

Inspecting the task

- *Task.analyzeproblem* – Analyze the data of a task.
- *Task.getnumcon* – Obtains the number of constraints.
- *Task.getnumcone* – Obtains the number of cones.
- *Task.getnumvar* – Obtains the number of variables.
- *Infrequent:* *Task.analyzesolution*, *Task.getacol*, *Task.getacolnumnz*, *Task.getacolslice*, *Task.getacolslicenumnz*, *Task.getacolslicetrip*, *Task.getaij*, *Task.getapiecenumnz*, *Task.getarow*, *Task.getarownumnz*, *Task.getarowslice*, *Task.getarowslicenumnz*, *Task.getarowslicetrip*, *Task.getbarablocktriplet*, *Task.getbaraidx*, *Task.getbaraidxij*, *Task.getbaraidxinfo*, *Task.getbarasparsity*, *Task.getbarcblocktriplet*, *Task.getbarcidx*, *Task.getbarcidxinfo*, *Task.getbarcidxj*, *Task.getbarcsparsity*, *Task.getbarvarname*, *Task.getbarvarnameindex*, *Task.getbarvarnamelen*, *Task.getc*, *Task.getcfix*, *Task.getcj*, *Task.getclist*, *Task.getconbound*, *Task.getconboundslice*,

Task.getcone, *Task.getconeinfo*, *Task.getconename*, *Task.getconenameindex*, *Task.getconenamelen*, *Task.getconname*, *Task.getconnameindex*, *Task.getconnamelen*, *Task.getcslice*, *Task.getdimbarvarj*, *Task.getlenbarvarj*, *Task.getmaxnumanz*, *Task.getmaxnumbarvar*, *Task.getmaxnumcon*, *Task.getmaxnumcone*, *Task.getmaxnumqnz*, *Task.getmaxnumvar*, *Task.getnumanz*, *Task.getnumanz64*, *Task.getnumbarablocktriplets*, *Task.getnumbaranz*, *Task.getnumbarcblocktriplets*, *Task.getnumbarcnz*, *Task.getnumbarvar*, *Task.getnumconemem*, *Task.getnumintvar*, *Task.getnumparam*, *Task.getnumqconknz*, *Task.getnumqobjnz*, *Task.getnumsymmat*, *Task.getobjname*, *Task.getobjnamelen*, *Task.getprobtype*, *Task.getqconk*, *Task.getqobj*, *Task.getqobjij*, *Task.getsparsesymmat*, *Task.getsymmatinfo*, *Task.gettaskname*, *Task.gettasknamelen*, *Task.getvarbound*, *Task.getvarboundslice*, *Task.getvarname*, *Task.getvarnameindex*, *Task.getvarnamelen*, *Task.getvartype*, *Task.getvartypelist*, *Task.readsummary*

License system

- *Env.checkoutlicense* – Check out a license feature from the license server ahead of time.
- *Env.putlicensedebug* – Enables debug information for the license system.
- *Env.putlicensepath* – Set the path to the license file.
- *Env.putlicensewait* – Control whether mosek should wait for an available license if no license is available.
- *Infrequent:* *Env.checkinall*, *Env.checkinlicense*, *Env.licensecleanup*, *Env.putlicensecode*

Linear algebra

- *Infrequent:* *Env.axpy*, *Env.computesparsecholesky*, *Env.dot*, *Env.gemm*, *Env.gemv*, *Env.potrf*, *Env.sparsetriangularsolvedense*, *Env.syeig*, *Env.syevd*, *Env.syrk*

Logging

- *Task.linkfiletostream* – Directs all output from a task stream to a file.
- *Task.onesolutionsummary* – Prints a short summary of a specified solution.
- *Task.optimizersummary* – Prints a short summary with optimizer statistics from last optimization.
- *Task.set_Stream* – Directs all output from a task stream to a callback function.
- *Task.solutionsummary* – Prints a short summary of the current solutions.
- *Infrequent:* *Env.echointro*, *Env.linkfiletostream*, *Env.set_Stream*

Names

- *Env.getcodedesc* – Obtains a short description of a response code.
- *Task.putbarvarname* – Sets the name of a semidefinite variable.
- *Task.putconename* – Sets the name of a cone.
- *Task.putconname* – Sets the name of a constraint.
- *Task.putobjname* – Assigns a new name to the objective.
- *Task.puttaskname* – Assigns a new name to the task.

- *Task.putvarname* – Sets the name of a variable.
- *Infrequent:* *Task.analyzenames*, *Task.generateconenames*, *Task.generateconnames*, *Task.generatevarnames*, *Task.getbarvarname*, *Task.getbarvarnameindex*, *Task.getbarvarnamelen*, *Task.getconename*, *Task.getconenameindex*, *Task.getconenamelen*, *Task.getconname*, *Task.getconnameindex*, *Task.getconnamelen*, *Task.getobjname*, *Task.getobjnamelen*, *Task.getstrparam*, *Task.getstrparamlen*, *Task.gettaskname*, *Task.gettasknamelen*, *Task.getvarname*, *Task.getvarnameindex*, *Task.getvarnamelen*, *Task.isdouparname*, *Task.isintparname*, *Task.isstrparname*, *Task.strtoconetype*, *Task.strtosk*

Optimization

- *Task.optimize* – Optimizes the problem.

Parameters

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

Problem data - bounds

- *Task.putconbound* – Changes the bound for one constraint.
- *Task.putconboundslice* – Changes the bounds for a slice of the constraints.
- *Task.putvarbound* – Changes the bounds for one variable.
- *Task.putvarboundslice* – Changes the bounds for a slice of the variables.
- *Infrequent:* *Task.chgconbound*, *Task.chgvarbound*, *Task.getconbound*, *Task.getconboundslice*, *Task.getvarbound*, *Task.getvarboundslice*, *Task.inputdata*, *Task.putconboundlist*, *Task.putconboundlistconst*, *Task.putconboundsliceconst*, *Task.putvarboundlist*, *Task.putvarboundlistconst*, *Task.putvarboundsliceconst*

Problem data - cones

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

Problem data - constraints

- *Task.appendcons* – Appends a number of constraints to the optimization task.
- *Task.getnumcon* – Obtains the number of constraints.
- *Task.putconbound* – Changes the bound for one constraint.
- *Task.putconboundslice* – Changes the bounds for a slice of the constraints.
- *Task.putconname* – Sets the name of a constraint.
- *Task.removecons* – Removes a number of constraints.
- *Infrequent:* *Task.chgconbound*, *Task.generateconnames*, *Task.getconbound*, *Task.getconboundslice*, *Task.getconname*, *Task.getconnameindex*, *Task.getconnamelen*, *Task.getmaxnumcon*, *Task.getnumqconknz*, *Task.getqconk*, *Task.inputdata*, *Task.putconboundlist*, *Task.putconboundlistconst*, *Task.putconboundsliceconst*, *Task.putmaxnumcon*

Problem data - linear part

- *Task.appendcons* – Appends a number of constraints to the optimization task.
- *Task.appendvars* – Appends a number of variables to the optimization task.
- *Task.getnumcon* – Obtains the number of constraints.
- *Task.putacol* – Replaces all elements in one column of the linear constraint matrix.
- *Task.putacolslice* – Replaces all elements in a sequence of columns the linear constraint matrix.
- *Task.putaij* – Changes a single value in the linear coefficient matrix.
- *Task.putaijlist* – Changes one or more coefficients in the linear constraint matrix.
- *Task.putarow* – Replaces all elements in one row of the linear constraint matrix.
- *Task.putarowslice* – Replaces all elements in several rows the linear constraint matrix.
- *Task.putcfix* – Replaces the fixed term in the objective.
- *Task.putcj* – Modifies one linear coefficient in the objective.
- *Task.putconbound* – Changes the bound for one constraint.
- *Task.putconboundslice* – Changes the bounds for a slice of the constraints.
- *Task.putconname* – Sets the name of a constraint.
- *Task.putcslice* – Modifies a slice of the linear objective coefficients.
- *Task.putobjname* – Assigns a new name to the objective.
- *Task.putobjsense* – Sets the objective sense.
- *Task.putvarbound* – Changes the bounds for one variable.
- *Task.putvarboundslice* – Changes the bounds for a slice of the variables.
- *Task.putvarname* – Sets the name of a variable.
- *Task.removecons* – Removes a number of constraints.
- *Task.removevars* – Removes a number of variables.

- *Infrequent:* *Task.chgconbound*, *Task.chgvarbound*, *Task.generateconnames*, *Task.generatevarnames*, *Task.getacol*, *Task.getacolnumnz*, *Task.getacolslice*, *Task.getacolslicenumnz*, *Task.getacolslicetrip*, *Task.getaij*, *Task.getapiecenumnz*, *Task.getarow*, *Task.getarownumnz*, *Task.getarowslice*, *Task.getarowslicenumnz*, *Task.getarowslicetrip*, *Task.getatruncatetol*, *Task.getc*, *Task.getcfix*, *Task.getcj*, *Task.getclist*, *Task.getconbound*, *Task.getconboundslice*, *Task.getconname*, *Task.getconnameindex*, *Task.getconnamelen*, *Task.getcslice*, *Task.getmaxnumanz*, *Task.getmaxnumcon*, *Task.getmaxnumvar*, *Task.getnumanz*, *Task.getnumanz64*, *Task.getobjsense*, *Task.getvarbound*, *Task.getvarboundslice*, *Task.getvarname*, *Task.getvarnameindex*, *Task.getvarnamelen*, *Task.inputdata*, *Task.putacollist*, *Task.putarowlist*, *Task.putatruncatetol*, *Task.putclist*, *Task.putconboundlist*, *Task.putconboundlistconst*, *Task.putconboundsliceconst*, *Task.putvarboundlist*, *Task.putvarboundlistconst*, *Task.putvarboundsliceconst*

Problem data - objective

- *Task.putbarcj* – Changes one element in barc.
- *Task.putcfix* – Replaces the fixed term in the objective.
- *Task.putcj* – Modifies one linear coefficient in the objective.
- *Task.putcslice* – Modifies a slice of the linear objective coefficients.
- *Task.putobjname* – Assigns a new name to the objective.
- *Task.putobjsense* – Sets the objective sense.
- *Task.putqobj* – Replaces all quadratic terms in the objective.
- *Task.putqobjij* – Replaces one coefficient in the quadratic term in the objective.
- *Infrequent:* *Task.putclist*

Problem data - quadratic part

- *Task.putqcon* – Replaces all quadratic terms in constraints.
- *Task.putqconk* – Replaces all quadratic terms in a single constraint.
- *Task.putqobj* – Replaces all quadratic terms in the objective.
- *Task.putqobjij* – Replaces one coefficient in the quadratic term in the objective.
- *Infrequent:* *Task.getmaxnumqnz*, *Task.getnumqconknz*, *Task.getnumqobjnz*, *Task.getqconk*, *Task.getqobj*, *Task.getqobjij*, *Task.putmaxnumqnz*, *Task.toconic*

Problem data - semidefinite

- *Task.appendbarvars* – Appends semidefinite variables to the problem.
- *Task.appendsparsesymmat* – Appends a general sparse symmetric matrix to the storage of symmetric matrices.
- *Task.appendsparsesymmatlist* – Appends a general sparse symmetric matrix to the storage of symmetric matrices.
- *Task.putbaraij* – Inputs an element of barA.
- *Task.putbaraijlist* – Inputs list of elements of barA.
- *Task.putbararowlist* – Replace a set of rows of barA

- *Task.putbarcj* – Changes one element in barc.
- *Task.putbarvarname* – Sets the name of a semidefinite variable.
- *Infrequent:* *Task.getbarablocktriplet*, *Task.getbaraidx*, *Task.getbaraidxij*, *Task.getbaraidxinfo*, *Task.getbarasparsity*, *Task.getbarcblocktriplet*, *Task.getbarcidx*, *Task.getbarcidxinfo*, *Task.getbarcidxj*, *Task.getbarcsparsity*, *Task.getdimbarvarj*, *Task.getlenbarvarj*, *Task.getmaxnumbarvar*, *Task.getnumbarablocktriplets*, *Task.getnumbaranz*, *Task.getnumbarcblocktriplets*, *Task.getnumbarcnz*, *Task.getnumbarvar*, *Task.getnumsymmat*, *Task.getsparsesymmat*, *Task.getsymmatinfo*, *Task.putbarablocktriplet*, *Task.putbarcblocktriplet*, *Task.putmaxnumanz*, *Task.putmaxnumbarvar*, *Task.removebarvars*

Problem data - variables

- *Task.appendvars* – Appends a number of variables to the optimization task.
- *Task.getnumvar* – Obtains the number of variables.
- *Task.putvarbound* – Changes the bounds for one variable.
- *Task.putvarboundslice* – Changes the bounds for a slice of the variables.
- *Task.putvarname* – Sets the name of a variable.
- *Task.putvartype* – Sets the variable type of one variable.
- *Task.removevars* – Removes a number of variables.
- *Infrequent:* *Task.chgvarbound*, *Task.generatevarnames*, *Task.getc*, *Task.getcj*, *Task.getmaxnumvar*, *Task.getnumintvar*, *Task.getvarbound*, *Task.getvarboundslice*, *Task.getvarname*, *Task.getvarnameindex*, *Task.getvarnamelen*, *Task.getvartype*, *Task.getvartypelist*, *Task.putclist*, *Task.putmaxnumvar*, *Task.putvarboundlist*, *Task.putvarboundlistconst*, *Task.putvarboundsliceconst*, *Task.putvartypelist*

Remote optimization

- *Task.asyncgetresult* – Request a response from a remote job.
- *Task.asyncoptimize* – Offload the optimization task to a solver server.
- *Task.asyncpoll* – Requests information about the status of the remote job.
- *Task.asyncstop* – Request that the job identified by the token is terminated.
- *Task.optimizermt* – Offload the optimization task to a solver server.

Responses, errors and warnings

- *Env.getcodedesc* – Obtains a short description of a response code.

Sensitivity analysis

- *Task.dualsensitivity* – Performs sensitivity analysis on objective coefficients.
- *Task.primalsensitivity* – Perform sensitivity analysis on bounds.
- *Task.sensitivityreport* – Creates a sensitivity report.

Solution - dual

- *Task.getdualobj* – Computes the dual objective value associated with the solution.
- *Task.gety* – Obtains the y vector for a solution.
- *Task.getyslice* – Obtains a slice of the y vector for a solution.
- *Infrequent:* *Task.getreducedcosts*, *Task.getslc*, *Task.getslcslice*, *Task.getslx*, *Task.getslxslice*, *Task.getsnx*, *Task.getsnxslice*, *Task.getsolution*, *Task.getsolutionslice*, *Task.getsuc*, *Task.getsucslice*, *Task.getsux*, *Task.getsuxslice*, *Task.putconsolutioni*, *Task.putslc*, *Task.putslcslice*, *Task.putslx*, *Task.putslxslice*, *Task.putsnx*, *Task.putsnxslice*, *Task.putsolution*, *Task.putsolutionyi*, *Task.putsuc*, *Task.putsucslice*, *Task.putsux*, *Task.putsuxslice*, *Task.putvarsolutionj*, *Task.putyslice*

Solution - primal

- *Task.getprimalobj* – Computes the primal objective value for the desired solution.
- *Task.getxx* – Obtains the xx vector for a solution.
- *Task.getxxslice* – Obtains a slice of the xx vector for a solution.
- *Task.putxx* – Sets the xx vector for a solution.
- *Task.putxxslice* – Sets a slice of the xx vector for a solution.
- *Infrequent:* *Task.getsolution*, *Task.getsolutionslice*, *Task.getxc*, *Task.getxcslice*, *Task.putconsolutioni*, *Task.putsolution*, *Task.putvarsolutionj*, *Task.putxc*, *Task.putxcslice*, *Task.puty*

Solution - semidefinite

- *Task.getbarsj* – Obtains the dual solution for a semidefinite variable.
- *Task.getbarsslice* – Obtains the dual solution for a sequence of semidefinite variables.
- *Task.getbarxj* – Obtains the primal solution for a semidefinite variable.
- *Task.getbarxslice* – Obtains the primal solution for a sequence of semidefinite variables.
- *Infrequent:* *Task.putbarsj*, *Task.putbarxj*

Solution information

- *Task.getdualobj* – Computes the dual objective value associated with the solution.
- *Task.getprimalobj* – Computes the primal objective value for the desired solution.
- *Task.getprosta* – Obtains the problem status.
- *Task.getpviolcon* – Computes the violation of a primal solution associated to a constraint.
- *Task.getpviolvar* – Computes the violation of a primal solution for a list of scalar variables.
- *Task.getsolsta* – Obtains the solution status.
- *Task.getsolutioninfo* – Obtains information about of a solution.
- *Task.onesolutionsummary* – Prints a short summary of a specified solution.
- *Task.solutiondef* – Checks whether a solution is defined.
- *Task.solutionsummary* – Prints a short summary of the current solutions.

- *Infrequent:* *Task.analyzesolution*, *Task.deletesolution*, *Task.getdualsolutionnorms*, *Task.getdviolbarvar*, *Task.getdviolcon*, *Task.getdviolcones*, *Task.getdviolvar*, *Task.getprimalsolutionnorms*, *Task.getpviolbarvar*, *Task.getpviolcones*, *Task.getskc*, *Task.getskcslice*, *Task.getskn*, *Task.getskx*, *Task.getskxslice*, *Task.getsolution*, *Task.getsolutionslice*, *Task.putconsolutioni*, *Task.putskc*, *Task.putskcslice*, *Task.putskx*, *Task.putskxslice*, *Task.putsolution*, *Task.putsolutionyi*, *Task.putvarsolutionj*

Solving systems with basis matrix

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

System, memory and debugging

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

Versions

- *Env.getversion* – Obtains MOSEK version information.

15.3 Class Env

mosek.Env

The **MOSEK** global environment.

Env.Env

```
Env(licensefile=None, debugfile=None)
```

Constructor of a new environment.

Parameters

- licensefile (str) – License file to use. (input)
- debugfile (str) – File where the memory debugging log is written. (input)

Env.Task

```
def Task (numcon, numvar) -> task
```

```
def Task () -> task
```

Creates a new task.

Parameters

- numcon (int) – An optional hint about the maximal number of constraints in the task. (input)
- numvar (int) – An optional hint about the maximal number of variables in the task. (input)

Return task (*Task*) – A new task.

Env.__del__

```
def __del__ ()
```

Free the underlying native allocation.

Env.axpy

```
def axpy (n, alpha, x, y)
```

Adds αx to y, i.e. performs the update

$$y := \alpha x + y.$$

Note that the result is stored overwriting y.

Parameters
- `n` (`int`) – Length of the vectors. (input)
- `alpha` (`float`) – The scalar that multiplies x. (input)
- `x` (`float[]`) – The x vector. (input)
- `y` (`float[]`) – The y vector. (input/output)

Groups *Linear algebra*

Env.checkinall

```
def checkinall ()
```

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

Groups *License system*

Env.checkinlicense

```
def checkinlicense (feature)
```

Check in a license feature to the license server. By default all licenses consumed by functions using a single environment are kept checked out for the lifetime of the **MOSEK** environment. This function checks in a given license feature back to the license server immediately.

If the given license feature is not checked out at all, or it is in use by a call to *`Task.optimize`*, calling this function has no effect.

Please note that returning a license to the license server incurs a small overhead, so frequent calls to this function should be avoided.

Parameters `feature` (*`mosek.feature`*) – Feature to check in to the license system. (input)

Groups *License system*

Env.checkoutlicense

```
def checkoutlicense (feature)
```

Checks out a license feature from the license server. Normally the required license features will be automatically checked out the first time they are needed by the function *`Task.optimize`*. This function can be used to check out one or more features ahead of time.

The feature will remain checked out until the environment is deleted or the function *`Env.checkinlicense`* is called.

If a given feature is already checked out when this function is called, the call has no effect.

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

Groups *License system*

`Env.computesparsecholesky`

```
def computesparsecholesky (multithread, ordermethod, tolsingular, anzc, aptrc, asubc,␣
↪avalc) -> perm, diag, lnzc, lptrc, lensubnval, lsubc, lvalc
```

The function computes a Cholesky factorization of a sparse positive semidefinite matrix. Sparsity is exploited during the computations to reduce the amount of space and work required. Both the input and output matrices are represented using the sparse format.

To be precise, given a symmetric matrix $A \in \mathbb{R}^{n\times n}$ the function computes a nonsingular lower triangular matrix L, a diagonal matrix D and a permutation matrix P such that

$$LL^T - D = PAP^T.$$

If `ordermethod` is zero then reordering heuristics are not employed and P is the identity.

If a pivot during the computation of the Cholesky factorization is less than

$$-\rho \cdot \max((PAP^T)_{jj}, 1.0)$$

then the matrix is declared negative semidefinite. On the hand if a pivot is smaller than

$$\rho \cdot \max((PAP^T)_{jj}, 1.0),$$

then D_{jj} is increased from zero to

$$\rho \cdot \max((PAP^T)_{jj}, 1.0).$$

Therefore, if A is sufficiently positive definite then D will be the zero matrix. Here ρ is set equal to value of `tolsingular`.

Parameters

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

Return

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

Groups *Linear algebra*

`Env.dot`

```
def dot (n, x, y) -> xty
```

Computes the inner product of two vectors x, y of length $n \geq 0$, i.e

$$x \cdot y = \sum_{i=1}^{n} x_i y_i.$$

Note that if $n = 0$, then the result of the operation is 0.

Parameters

- `n` (`int`) – Length of the vectors. (input)
- `x` (`float[]`) – The x vector. (input)
- `y` (`float[]`) – The y vector. (input)

Return `xty` (`float`) – The result of the inner product between x and y.

Groups *Linear algebra*

`Env.echointro`

```
def echointro (longver)
```

Prints an intro to message stream.

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

Groups *Logging*

`Env.gemm`

```
def gemm (transa, transb, m, n, k, alpha, a, b, beta, c)
```

Performs a matrix multiplication plus addition of dense matrices. Given A, B and C of compatible dimensions, this function computes

$$C := \alpha op(A) op(B) + \beta C$$

where α, β are two scalar values. The function $op(X)$ denotes X if transX is *transpose.no*, or X^T if set to *transpose.yes*. The matrix C has m rows and n columns, and the other matrices must have compatible dimensions.

The result of this operation is stored in C.

Parameters

- `transa` (*mosek.transpose*) – Indicates whether the matrix A must be transposed. (input)
- `transb` (*mosek.transpose*) – Indicates whether the matrix B must be transposed. (input)
- `m` (`int`) – Indicates the number of rows of matrix C. (input)
- `n` (`int`) – Indicates the number of columns of matrix C. (input)
- `k` (`int`) – Specifies the common dimension along which $op(A)$ and $op(B)$ are multiplied. For example, if neither A nor B are transposed, then this is the number of columns in A and also the number of rows in B. (input)
- `alpha` (`float`) – A scalar value multiplying the result of the matrix multiplication. (input)
- `a` (`float[]`) – The pointer to the array storing matrix A in a column-major format. (input)
- `b` (`float[]`) – The pointer to the array storing matrix B in a column-major format. (input)
- `beta` (`float`) – A scalar value that multiplies C. (input)

- c (float[]) – The pointer to the array storing matrix C in a column-major format. (input/output)

Groups *Linear algebra*

Env.gemv

```
def gemv (transa, m, n, alpha, a, x, beta, y)
```

Computes the multiplication of a scaled dense matrix times a dense vector, plus a scaled dense vector. Precisely, if trans is *transpose.no* then the update is

$$y := \alpha Ax + \beta y,$$

and if trans is *transpose.yes* then

$$y := \alpha A^T x + \beta y,$$

where α, β are scalar values and A is a matrix with m rows and n columns.

Note that the result is stored overwriting y.

Parameters

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

Groups *Linear algebra*

Env.getcodedesc

```
@staticmethod
def getcodedesc (code) -> symname, str
```

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

Parameters code (*mosek.rescode*) – A valid **MOSEK** response code. (input)

Return

- symname (str) – Symbolic name corresponding to code.
- str (str) – Obtains a short description of a response code.

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

Env.getversion

```
@staticmethod
def getversion () -> major, minor, revision
```

Obtains **MOSEK** version information.

Return

- major (int) – Major version number.
- minor (int) – Minor version number.

- revision (int) – Revision number.

Groups *Versions*

Env.licensecleanup

```
@staticmethod
def licensecleanup ()
```

Stops all threads and deletes all handles used by the license system. If this function is called, it must be called as the last **MOSEK** API call. No other **MOSEK** API calls are valid after this.

Groups *License system*

Env.linkfiletostream

```
def linkfiletostream (whichstream, filename, append)
```

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

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

Groups *Logging*

Env.potrf

```
def potrf (uplo, n, a)
```

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

Parameters
- uplo (*mosek.uplo*) – Indicates whether the upper or lower triangular part of the matrix is stored. (input)
- n (int) – Dimension of the symmetric matrix. (input)
- a (float[]) – A symmetric matrix stored in column-major order. Only the lower or the upper triangular part is used, accordingly with the `uplo` parameter. It will contain the result on exit. (input/output)

Groups *Linear algebra*

Env.putlicensecode

```
def putlicensecode (code)
```

Input a runtime license code.

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

Groups *License system*

Env.putlicensedebug

```
def putlicensedebug (licdebug)
```

Enables debug information for the license system. If `licdebug` is non-zero, then **MOSEK** will print debug info regarding the license checkout.

Parameters `licdebug` (`int`) – Whether license checkout debug info should be printed. (input)

Groups *License system*

`Env.putlicensepath`

```
def putlicensepath (licensepath)
```

Set the path to the license file.

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

Groups *License system*

`Env.putlicensewait`

```
def putlicensewait (licwait)
```

Control whether **MOSEK** should wait for an available license if no license is available. If `licwait` is non-zero, then **MOSEK** will wait for `licwait-1` milliseconds between each check for an available license.

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

Groups *License system*

`Env.set_Stream`

```
def set_Stream (whichstream, callback)
```

Directs all output from a environment stream to a callback function.

Parameters

- `whichstream` (*`streamtype`*) – Index of the stream. (input)
- `callback` (*`streamfunc`*) – The callback function. (input)

`Env.setupthreads`

```
def setupthreads (numthreads)
```

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

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

Groups *System, memory and debugging*

`Env.sparsetriangularsolvedense`

```
def sparsetriangularsolvedense (transposed, lnzc, lptrc, lsubc, lvalc, b)
```

The function solves a triangular system of the form

$$Lx = b$$

or

$$L^T x = b$$

where L is a sparse lower triangular nonsingular matrix. This implies in particular that diagonals in L are nonzero.

Parameters

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

Groups *Linear algebra*

`Env.syeig`

```
def syeig (uplo, n, a, w)
```

Computes all eigenvalues of a real symmetric matrix A. Given a matrix $A \in \mathbb{R}^{n \times n}$ it returns a vector $w \in \mathbb{R}^n$ containing the eigenvalues of A.

Parameters

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

Groups *Linear algebra*

`Env.syevd`

```
def syevd (uplo, n, a, w)
```

Computes all the eigenvalues and eigenvectors a real symmetric matrix. Given the input matrix $A \in \mathbb{R}^{n \times n}$, this function returns a vector $w \in \mathbb{R}^n$ containing the eigenvalues of A and it also computes the eigenvectors of A. Therefore, this function computes the eigenvalue decomposition of A as

$$A = UVU^T,$$

where $V = \mathbf{diag}(w)$ and U contains the eigenvectors of A.

Note that the matrix U overwrites the input data A.

Parameters

- `uplo` (*mosek.uplo*) – Indicates whether the upper or lower triangular part is used. (input)
- `n` (`int`) – Dimension of the symmetric input matrix. (input)
- `a` (`float[]`) – A symmetric matrix A stored in column-major order. Only the part indicated by `uplo` is used. On exit it will be overwritten by the matrix U. (input/output)
- `w` (`float[]`) – Array of length at least `n` containing the eigenvalues of A. (output)

Groups *Linear algebra*

`Env.syrk`

```
def syrk (uplo, trans, n, k, alpha, a, beta, c)
```

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

Given a symmetric matrix $C \in \mathbb{R}^{n \times n}$, two scalars α, β and a matrix A of rank $k \leq n$, it computes either

$$C := \alpha A A^T + \beta C,$$

when `trans` is set to *transpose.no* and $A \in \mathbb{R}^{n \times k}$, or

$$C := \alpha A^T A + \beta C,$$

when `trans` is set to *transpose.yes* and $A \in \mathbb{R}^{k \times n}$.

Only the part of C indicated by `uplo` is used and only that part is updated with the result.

Parameters

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

Groups *Linear algebra*

15.4 Class Task

`mosek.Task`

Represents an optimization task.

`Task.Task`

```
Task(env)
```

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

```
Task(other)
```

Constructor of a new optimization task.

Parameters

- `env` (*Env*) – Parent environment. (input)
- `numcon` (`int`) – An optional hint about the maximal number of constraints in the task. (input)
- `numvar` (`int`) – An optional hint about the maximal number of variables in the task. (input)
- `other` (*Task*) – A task that will be cloned. (input)

`Task.__del__`

```
def __del__ ()
```

Free the underlying native allocation.

Task.analyzenames

```
def analyzenames (whichstream, nametype)
```

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

Parameters
- whichstream (*mosek.streamtype*) – Index of the stream. (input)
- nametype (*mosek.nametype*) – The type of names e.g. valid in MPS or LP files. (input)

Groups *Names*

Task.analyzeproblem

```
def analyzeproblem (whichstream)
```

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

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

Groups *Inspecting the task*

Task.analyzesolution

```
def analyzesolution (whichstream, whichsol)
```

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

By default this function prints information about the largest infeasibilites in the solution, the primal (and possibly dual) objective value and the solution status.

Following parameters can be used to configure the printed statistics:

- *iparam.ana_sol_basis* enables or disables printing of statistics specific to the basis solution (condition number, number of basic variables etc.). Default is on.
- *iparam.ana_sol_print_violated* enables or disables listing names of all constraints (both primal and dual) which are violated by the solution. Default is off.
- *dparam.ana_sol_infeas_tol* is the tolerance defining when a constraint is considered violated. If a constraint is violated more than this, it will be listed in the summary.

Parameters
- whichstream (*mosek.streamtype*) – Index of the stream. (input)
- whichsol (*mosek.soltype*) – Selects a solution. (input)

Groups *Solution information*, *Inspecting the task*

Task.appendbarvars

```
def appendbarvars (dim)
```

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

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

Groups *Problem data - semidefinite*

Task.appendcone

```
def appendcone (ct, conepar, submem)
```

Appends a new conic constraint to the problem. Hence, add a constraint

$$\hat{x} \in \mathcal{K}$$

to the problem, where $\mathcal{K}$ is a convex cone. $\hat{x}$ is a subset of the variables which will be specified by the argument `submem`. Cone type is specified by `ct`.

Define

$$\hat{x} = x_{\texttt{submem}[0]}, \ldots, x_{\texttt{submem}[\texttt{nummem}-1]}.$$

Depending on the value of `ct` this function appends one of the constraints:

- Quadratic cone (*conetype.quad*, requires `nummem` ≥ 1):

$$\hat{x}_0 \geq \sqrt{\sum_{i=1}^{i<\texttt{nummem}} \hat{x}_i^2}$$

- Rotated quadratic cone (*conetype.rquad*, requires `nummem` ≥ 2):

$$2\hat{x}_0\hat{x}_1 \geq \sum_{i=2}^{i<\texttt{nummem}} \hat{x}_i^2, \quad \hat{x}_0, \hat{x}_1 \geq 0$$

- Primal exponential cone (*conetype.pexp*, requires `nummem` $= 3$):

$$\hat{x}_0 \geq \hat{x}_1 \exp(\hat{x}_2/\hat{x}_1), \quad \hat{x}_0, \hat{x}_1 \geq 0$$

- Primal power cone (*conetype.ppow*, requires `nummem` ≥ 2):

$$\hat{x}_0^{\alpha}\hat{x}_1^{1-\alpha} \geq \sqrt{\sum_{i=2}^{i<\texttt{nummem}} \hat{x}_i^2}, \quad \hat{x}_0, \hat{x}_1 \geq 0$$

  where α is the cone parameter specified by `conepar`.
- Dual exponential cone (*conetype.dexp*, requires `nummem` $= 3$):

$$\hat{x}_0 \geq -\hat{x}_2 e^{-1} \exp(\hat{x}_1/\hat{x}_2), \quad \hat{x}_2 \leq 0, \hat{x}_0 \geq 0$$

- Dual power cone (*conetype.dpow*, requires `nummem` ≥ 2):

$$\left(\frac{\hat{x}_0}{\alpha}\right)^{\alpha} \left(\frac{\hat{x}_1}{1-\alpha}\right)^{1-\alpha} \geq \sqrt{\sum_{i=2}^{i<\texttt{nummem}} \hat{x}_i^2}, \quad \hat{x}_0, \hat{x}_1 \geq 0$$

  where α is the cone parameter specified by `conepar`.
- Zero cone (*conetype.zero*):

$$\hat{x}_i = 0 \text{ for all } i$$

Please note that the sets of variables appearing in different conic constraints must be disjoint.

For an explained code example see Sec. 6.3, Sec. 6.5 or Sec. 6.4.

Parameters

- `ct` (*mosek.conetype*) – Specifies the type of the cone. (input)
- `conepar` (`float`) – For the power cone it denotes the exponent alpha. For other cone types it is unused and can be set to 0. (input)
- `submem` (`int[]`) – Variable subscripts of the members in the cone. (input)

Groups *Problem data - cones*

`Task.appendconeseq`

```
def appendconeseq (ct, conepar, nummem, j)
```

Appends a new conic constraint to the problem, as in *Task.appendcone*. The function assumes the members of cone are sequential where the first member has index `j` and the last `j+nummem-1`.

Parameters

- `ct` (*mosek.conetype*) – Specifies the type of the cone. (input)
- `conepar` (`float`) – For the power cone it denotes the exponent alpha. For other cone types it is unused and can be set to 0. (input)
- `nummem` (`int`) – Number of member variables in the cone. (input)
- `j` (`int`) – Index of the first variable in the conic constraint. (input)

Groups *Problem data - cones*

`Task.appendconesseq`

```
def appendconesseq (ct, conepar, nummem, j)
```

Appends a number of conic constraints to the problem, as in *Task.appendcone*. The kth cone is assumed to be of dimension `nummem[k]`. Moreover, it is assumed that the first variable of the first cone has index j and starting from there the sequentially following variables belong to the first cone, then to the second cone and so on.

Parameters

- `ct` (*mosek.conetype* `[]`) – Specifies the type of the cone. (input)
- `conepar` (`float[]`) – For the power cone it denotes the exponent alpha. For other cone types it is unused and can be set to 0. (input)
- `nummem` (`int[]`) – Numbers of member variables in the cones. (input)
- `j` (`int`) – Index of the first variable in the first cone to be appended. (input)

Groups *Problem data - cones*

`Task.appendcons`

```
def appendcons (num)
```

Appends a number of constraints to the model. Appended constraints will be declared free. Please note that **MOSEK** will automatically expand the problem dimension to accommodate the additional constraints.

Parameters `num` (`int`) – Number of constraints which should be appended. (input)

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

`Task.appendsparsesymmat`

```
def appendsparsesymmat (dim, subi, subj, valij) -> idx
```

MOSEK maintains a storage of symmetric data matrices that is used to build $\overline{C}$ and $\overline{A}$. The storage can be thought of as a vector of symmetric matrices denoted E. Hence, E_i is a symmetric matrix of certain dimension.

This function appends a general sparse symmetric matrix on triplet form to the vector E of symmetric matrices. The vectors `subi`, `subj`, and `valij` contains the row subscripts, column subscripts and values of each element in the symmetric matrix to be appended. Since the matrix that is appended is symmetric, only the lower triangular part should be specified. Moreover, duplicates are not allowed.

Observe the function reports the index (position) of the appended matrix in E. This index should be used for later references to the appended matrix.

Parameters

- `dim` (`int`) – Dimension of the symmetric matrix that is appended. (input)
- `subi` (`int[]`) – Row subscript in the triplets. (input)
- `subj` (`int[]`) – Column subscripts in the triplets. (input)
- `valij` (`float[]`) – Values of each triplet. (input)

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

Groups *Problem data - semidefinite*

`Task.appendsparsesymmatlist`

```
def appendsparsesymmatlist (dims, nz, subi, subj, valij, idx)
```

MOSEK maintains a storage of symmetric data matrices that is used to build $\overline{C}$ and $\overline{A}$. The storage can be thought of as a vector of symmetric matrices denoted E. Hence, E_i is a symmetric matrix of certain dimension.

This function appends general sparse symmetric matrixes on triplet form to the vector E of symmetric matrices. The vectors `subi`, `subj`, and `valij` contains the row subscripts, column subscripts and values of each element in the symmetric matrix to be appended. Since the matrix that is appended is symmetric, only the lower triangular part should be specified. Moreover, duplicates are not allowed.

Observe the function reports the index (position) of the appended matrix in E. This index should be used for later references to the appended matrix.

Parameters

- `dims` (`int[]`) – Dimensions of the symmetric matrixes. (input)
- `nz` (`int[]`) – Number of nonzeros for each matrix. (input)
- `subi` (`int[]`) – Row subscript in the triplets. (input)
- `subj` (`int[]`) – Column subscripts in the triplets. (input)
- `valij` (`float[]`) – Values of each triplet. (input)
- `idx` (`int[]`) – Unique index assigned to the inputted matrix that can be used for later reference. (output)

Groups *Problem data - semidefinite*

`Task.appendvars`

```
def appendvars (num)
```

Appends a number of variables to the model. Appended variables will be fixed at zero. Please note that **MOSEK** will automatically expand the problem dimension to accommodate the additional variables.

Parameters `num` (`int`) – Number of variables which should be appended. (input)

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

`Task.asyncgetresult`

```
def asyncgetresult (server, port, token) -> respavailable, resp, trm
```

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

Parameters

- `server` (`str`) – Name or IP address of the solver server. (input)
- `port` (`str`) – Network port of the solver service. (input)
- `token` (`str`) – The task token. (input)

Return

- respavailable (int) – Indicates if a remote response is available. If this is not true, resp and trm should be ignored.
- resp (*mosek.rescode*) – Is the response code from the remote solver.
- trm (*mosek.rescode*) – Is either *rescode.ok* or a termination response code.

Groups *Remote optimization*

Task.asyncoptimize

```
def asyncoptimize (server, port) -> token
```

Offload the optimization task to a solver server defined by server:port. The call will return immediately and not wait for the result.

If the string parameter *sparam.remote_access_token* is not blank, it will be passed to the server as authentication.

Parameters

- server (str) – Name or IP address of the solver server (input)
- port (str) – Network port of the solver service (input)

Return token (str) – Returns the task token

Groups *Remote optimization*

Task.asyncpoll

```
def asyncpoll (server, port, token) -> respavailable, resp, trm
```

Requests information about the status of the remote job.

Parameters

- server (str) – Name or IP address of the solver server (input)
- port (str) – Network port of the solver service (input)
- token (str) – The task token (input)

Return

- respavailable (int) – Indicates if a remote response is available. If this is not true, resp and trm should be ignored.
- resp (*mosek.rescode*) – Is the response code from the remote solver.
- trm (*mosek.rescode*) – Is either *rescode.ok* or a termination response code.

Groups *Remote optimization*

Task.asyncstop

```
def asyncstop (server, port, token)
```

Request that the job identified by the token is terminated.

Parameters

- server (str) – Name or IP address of the solver server (input)
- port (str) – Network port of the solver service (input)
- token (str) – The task token (input)

Groups *Remote optimization*

Task.basiscond

```
def basiscond () -> nrmbasis, nrminvbasis
```

If a basic solution is available and it defines a nonsingular basis, then this function computes the 1-norm estimate of the basis matrix and a 1-norm estimate for the inverse of the basis matrix. The 1-norm estimates are computed using the method outlined in *[Ste98]*, pp. 388-391.

By definition the 1-norm condition number of a matrix B is defined as

$$\kappa_1(B) := \|B\|_1\|B^{-1}\|_1.$$

Moreover, the larger the condition number is the harder it is to solve linear equation systems involving B. Given estimates for $\|B\|_1$ and $\|B^{-1}\|_1$ it is also possible to estimate $\kappa_1(B)$.

Return

- `nrmbasis` (`float`) – An estimate for the 1-norm of the basis.
- `nrminvbasis` (`float`) – An estimate for the 1-norm of the inverse of the basis.

Groups *Solving systems with basis matrix*

`Task.checkmem`

```
def checkmem (file, line)
```

Checks the memory allocated by the task.

Parameters

- `file` (`str`) – File from which the function is called. (input)
- `line` (`int`) – Line in the file from which the function is called. (input)

Groups *System, memory and debugging*

`Task.chgconbound`

```
def chgconbound (i, lower, finite, value)
```

Changes a bound for one constraint.

If `lower` is non-zero, then the lower bound is changed as follows:

$$\text{new lower bound} = \begin{cases} -\infty, & \texttt{finite} = 0, \\ \texttt{value} & \text{otherwise.} \end{cases}$$

Otherwise if `lower` is zero, then

$$\text{new upper bound} = \begin{cases} \infty, & \texttt{finite} = 0, \\ \texttt{value} & \text{otherwise.} \end{cases}$$

Please note that this function automatically updates the bound key for the bound, in particular, if the lower and upper bounds are identical, the bound key is changed to `fixed`.

Parameters

- `i` (`int`) – Index of the constraint for which the bounds should be changed. (input)
- `lower` (`int`) – If non-zero, then the lower bound is changed, otherwise the upper bound is changed. (input)
- `finite` (`int`) – If non-zero, then `value` is assumed to be finite. (input)
- `value` (`float`) – New value for the bound. (input)

Groups *Problem data - bounds*, *Problem data - constraints*, *Problem data - linear part*

`Task.chgvarbound`

```
def chgvarbound (j, lower, finite, value)
```

Changes a bound for one variable.

If `lower` is non-zero, then the lower bound is changed as follows:

$$\text{new lower bound} = \begin{cases} -\infty, & \texttt{finite} = 0, \\ \texttt{value} & \text{otherwise.} \end{cases}$$

Otherwise if `lower` is zero, then

$$\text{new upper bound} = \begin{cases} \infty, & \texttt{finite} = 0, \\ \texttt{value} & \text{otherwise.} \end{cases}$$

Please note that this function automatically updates the bound key for the bound, in particular, if the lower and upper bounds are identical, the bound key is changed to `fixed`.

Parameters

- `j` (`int`) – Index of the variable for which the bounds should be changed. (input)
- `lower` (`int`) – If non-zero, then the lower bound is changed, otherwise the upper bound is changed. (input)
- `finite` (`int`) – If non-zero, then `value` is assumed to be finite. (input)
- `value` (`float`) – New value for the bound. (input)

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

`Task.commitchanges`

```
def commitchanges ()
```

Commits all cached problem changes to the task. It is usually not necessary to call this function explicitly since changes will be committed automatically when required.

Groups *Environment and task management*

`Task.deletesolution`

```
def deletesolution (whichsol)
```

Undefine a solution and free the memory it uses.

Parameters `whichsol` (*`mosek.soltype`*) – Selects a solution. (input)

Groups *Environment and task management*, *Solution information*

`Task.dualsensitivity`

```
def dualsensitivity (subj, leftpricej, rightpricej, leftrangej, rightrangej)
```

Calculates sensitivity information for objective coefficients. The indexes of the coefficients to analyze are

$$\{\texttt{subj}[i] \mid i = 0, \ldots, \texttt{numj} - 1\}$$

The type of sensitivity analysis to perform (basis or optimal partition) is controlled by the parameter *`iparam.sensitivity_type`*.

For an example, please see Section *Example: Sensitivity Analysis*.

Parameters

- `subj` (`int[]`) – Indexes of objective coefficients to analyze. (input)
- `leftpricej` (`float[]`) – `leftpricej`$[j]$ is the left shadow price for the coefficient with index `subj`[j]. (output)

- rightpricej (float[]) – rightpricej[j] is the right shadow price for the coefficient with index subj[j]. (output)
- leftrangej (float[]) – leftrangej[j] is the left range β_1 for the coefficient with index subj[j]. (output)
- rightrangej (float[]) – rightrangej[j] is the right range β_2 for the coefficient with index subj[j]. (output)

Groups *Sensitivity analysis*

Task.generateconenames

```
def generateconenames (subk, fmt, dims, sp)
```

Generates systematic names for cone.

Parameters

- subk (int[]) – Indexes of the cone. (input)
- fmt (str) – The cone name formatting string. (input)
- dims (int[]) – Dimensions in the shape. (input)
- sp (int[]) – Items that should be named. (input)

Groups *Names*, *Problem data - cones*

Task.generateconnames

```
def generateconnames (subi, fmt, dims, sp)
```

Generates systematic names for constraints.

Parameters

- subi (int[]) – Indexes of the constraints. (input)
- fmt (str) – The constraint name formatting string. (input)
- dims (int[]) – Dimensions in the shape. (input)
- sp (int[]) – Items that should be named. (input)

Groups *Names*, *Problem data - constraints*, *Problem data - linear part*

Task.generatevarnames

```
def generatevarnames (subj, fmt, dims, sp)
```

Generates systematic names for variables.

Parameters

- subj (int[]) – Indexes of the variables. (input)
- fmt (str) – The variable name formatting string. (input)
- dims (int[]) – Dimensions in the shape. (input)
- sp (int[]) – Items that should be named. (input)

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

Task.getacol

```
def getacol (j, subj, valj) -> nzj
```

Obtains one column of A in a sparse format.

Parameters

- j (int) – Index of the column. (input)
- subj (int[]) – Row indices of the non-zeros in the column obtained. (output)

- valj (float[]) – Numerical values in the column obtained. (output)

Return nzj (int) – Number of non-zeros in the column obtained.

Groups *Problem data - linear part*, *Inspecting the task*

Task.getacolnumnz

```
def getacolnumnz (i) -> nzj
```

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

Parameters i (int) – Index of the column. (input)

Return nzj (int) – Number of non-zeros in the j-th column of A.

Groups *Problem data - linear part*, *Inspecting the task*

Task.getacolslice

```
def getacolslice (first, last, ptrb, ptre, sub, val)
```

Obtains a sequence of columns from A in sparse format.

Parameters

- first (int) – Index of the first column in the sequence. (input)
- last (int) – Index of the last column in the sequence **plus one**. (input)
- ptrb (int[]) – ptrb[t] is an index pointing to the first element in the t-th column obtained. (output)
- ptre (int[]) – ptre[t] is an index pointing to the last element plus one in the t-th column obtained. (output)
- sub (int[]) – Contains the row subscripts. (output)
- val (float[]) – Contains the coefficient values. (output)

Groups *Problem data - linear part*, *Inspecting the task*

Task.getacolslicenumnz

```
def getacolslicenumnz (first, last) -> numnz
```

Obtains the number of non-zeros in a slice of columns of A.

Parameters

- first (int) – Index of the first column in the sequence. (input)
- last (int) – Index of the last column **plus one** in the sequence. (input)

Return numnz (int) – Number of non-zeros in the slice.

Groups *Problem data - linear part*, *Inspecting the task*

Task.getacolslicetrip

```
def getacolslicetrip (first, last, subi, subj, val)
```

Obtains a sequence of columns from A in sparse triplet format. The function returns the content of all columns whose index j satisfies first <= j < last. The triplets corresponding to nonzero entries are stored in the arrays subi, subj and val.

Parameters

- first (int) – Index of the first column in the sequence. (input)
- last (int) – Index of the last column in the sequence **plus one**. (input)
- subi (int[]) – Constraint subscripts. (output)
- subj (int[]) – Column subscripts. (output)

- val (float[]) – Values. (output)

Groups *Problem data - linear part*, *Inspecting the task*

Task.getaij

```
def getaij (i, j) -> aij
```

Obtains a single coefficient in A.

Parameters
- i (int) – Row index of the coefficient to be returned. (input)
- j (int) – Column index of the coefficient to be returned. (input)

Return aij (float) – The required coefficient $a_{i,j}$.

Groups *Problem data - linear part*, *Inspecting the task*

Task.getapiecenumnz

```
def getapiecenumnz (firsti, lasti, firstj, lastj) -> numnz
```

Obtains the number non-zeros in a rectangular piece of A, i.e. the number of elements in the set

$$\{(i,j)\ :\ a_{i,j} \neq 0,\ \mathtt{firsti} \leq i \leq \mathtt{lasti}-1,\ \mathtt{firstj} \leq j \leq \mathtt{lastj}-1\}$$

This function is not an efficient way to obtain the number of non-zeros in one row or column. In that case use the function *Task.getarownumnz* or *Task.getacolnumnz*.

Parameters
- firsti (int) – Index of the first row in the rectangular piece. (input)
- lasti (int) – Index of the last row plus one in the rectangular piece. (input)
- firstj (int) – Index of the first column in the rectangular piece. (input)
- lastj (int) – Index of the last column plus one in the rectangular piece. (input)

Return numnz (int) – Number of non-zero A elements in the rectangular piece.

Groups *Problem data - linear part*, *Inspecting the task*

Task.getarow

```
def getarow (i, subi, vali) -> nzi
```

Obtains one row of A in a sparse format.

Parameters
- i (int) – Index of the row. (input)
- subi (int[]) – Column indices of the non-zeros in the row obtained. (output)
- vali (float[]) – Numerical values of the row obtained. (output)

Return nzi (int) – Number of non-zeros in the row obtained.

Groups *Problem data - linear part*, *Inspecting the task*

Task.getarownumnz

```
def getarownumnz (i) -> nzi
```

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

Parameters i (int) – Index of the row. (input)

Return nzi (int) – Number of non-zeros in the i-th row of A.

Groups *Problem data - linear part*, *Inspecting the task*

`Task.getarowslice`

```
def getarowslice (first, last, ptrb, ptre, sub, val)
```

Obtains a sequence of rows from A in sparse format.

Parameters
- `first` (`int`) – Index of the first row in the sequence. (input)
- `last` (`int`) – Index of the last row in the sequence **plus one**. (input)
- `ptrb` (`int[]`) – `ptrb[t]` is an index pointing to the first element in the t-th row obtained. (output)
- `ptre` (`int[]`) – `ptre[t]` is an index pointing to the last element plus one in the t-th row obtained. (output)
- `sub` (`int[]`) – Contains the column subscripts. (output)
- `val` (`float[]`) – Contains the coefficient values. (output)

Groups *Problem data - linear part*, *Inspecting the task*

`Task.getarowslicenumnz`

```
def getarowslicenumnz (first, last) -> numnz
```

Obtains the number of non-zeros in a slice of rows of A.

Parameters
- `first` (`int`) – Index of the first row in the sequence. (input)
- `last` (`int`) – Index of the last row **plus one** in the sequence. (input)

Return `numnz` (`int`) – Number of non-zeros in the slice.

Groups *Problem data - linear part*, *Inspecting the task*

`Task.getarowslicetrip`

```
def getarowslicetrip (first, last, subi, subj, val)
```

Obtains a sequence of rows from A in sparse triplet format. The function returns the content of all rows whose index `i` satisfies `first <= i < last`. The triplets corresponding to nonzero entries are stored in the arrays `subi`, `subj` and `val`.

Parameters
- `first` (`int`) – Index of the first row in the sequence. (input)
- `last` (`int`) – Index of the last row in the sequence **plus one**. (input)
- `subi` (`int[]`) – Constraint subscripts. (output)
- `subj` (`int[]`) – Column subscripts. (output)
- `val` (`float[]`) – Values. (output)

Groups *Problem data - linear part*, *Inspecting the task*

`Task.getatruncatetol`

```
def getatruncatetol (tolzero)
```

Obtains the tolerance value set with *`Task.putatruncatetol`*.

Parameters `tolzero` (`float[]`) – All elements $|a_{i,j}|$ less than this tolerance is truncated to zero. (output)

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

`Task.getbarablocktriplet`

```
def getbarablocktriplet (subi, subj, subk, subl, valijkl) -> num
```

Obtains $\overline{A}$ in block triplet form.

Parameters
- `subi` (`int[]`) – Constraint index. (output)
- `subj` (`int[]`) – Symmetric matrix variable index. (output)
- `subk` (`int[]`) – Block row index. (output)
- `subl` (`int[]`) – Block column index. (output)
- `valijkl` (`float[]`) – The numerical value associated with each block triplet. (output)

Return `num` (`int`) – Number of elements in the block triplet form.

Groups *Problem data - semidefinite*, *Inspecting the task*

`Task.getbaraidx`

```
def getbaraidx (idx, sub, weights) -> i, j, num
```

Obtains information about an element in $\overline{A}$. Since $\overline{A}$ is a sparse matrix of symmetric matrices, only the nonzero elements in $\overline{A}$ are stored in order to save space. Now $\overline{A}$ is stored vectorized i.e. as one long vector. This function makes it possible to obtain information such as the row index and the column index of a particular element of the vectorized form of $\overline{A}$.

Please observe if one element of $\overline{A}$ is inputted multiple times then it may be stored several times in vectorized form. In that case the element with the highest index is the one that is used.

Parameters
- `idx` (`int`) – Position of the element in the vectorized form. (input)
- `sub` (`int[]`) – A list indexes of the elements from symmetric matrix storage that appear in the weighted sum. (output)
- `weights` (`float[]`) – The weights associated with each term in the weighted sum. (output)

Return
- `i` (`int`) – Row index of the element at position `idx`.
- `j` (`int`) – Column index of the element at position `idx`.
- `num` (`int`) – Number of terms in weighted sum that forms the element.

Groups *Problem data - semidefinite*, *Inspecting the task*

`Task.getbaraidxij`

```
def getbaraidxij (idx) -> i, j
```

Obtains information about an element in $\overline{A}$. Since $\overline{A}$ is a sparse matrix of symmetric matrices, only the nonzero elements in $\overline{A}$ are stored in order to save space. Now $\overline{A}$ is stored vectorized i.e. as one long vector. This function makes it possible to obtain information such as the row index and the column index of a particular element of the vectorized form of $\overline{A}$.

Please note that if one element of $\overline{A}$ is inputted multiple times then it may be stored several times in vectorized form. In that case the element with the highest index is the one that is used.

Parameters `idx` (`int`) – Position of the element in the vectorized form. (input)

Return
- `i` (`int`) – Row index of the element at position `idx`.
- `j` (`int`) – Column index of the element at position `idx`.

Groups *Problem data - semidefinite*, *Inspecting the task*

Task.getbaraidxinfo

```
def getbaraidxinfo (idx) -> num
```

Each nonzero element in $\overline{A}_{ij}$ is formed as a weighted sum of symmetric matrices. Using this function the number of terms in the weighted sum can be obtained. See description of *Task.appendsparsesymmat* for details about the weighted sum.

Parameters `idx` (`int`) – The internal position of the element for which information should be obtained. (input)

Return `num` (`int`) – Number of terms in the weighted sum that form the specified element in $\overline{A}$.

Groups *Problem data - semidefinite*, *Inspecting the task*

Task.getbarasparsity

```
def getbarasparsity (idxij) -> numnz
```

The matrix $\overline{A}$ is assumed to be a sparse matrix of symmetric matrices. This implies that many of the elements in $\overline{A}$ are likely to be zero matrices. Therefore, in order to save space, only nonzero elements in $\overline{A}$ are stored on vectorized form. This function is used to obtain the sparsity pattern of $\overline{A}$ and the position of each nonzero element in the vectorized form of $\overline{A}$. From the index detailed information about each nonzero $\overline{A}_{i,j}$ can be obtained using *Task.getbaraidxinfo* and *Task.getbaraidx*.

Parameters `idxij` (`int[]`) – Position of each nonzero element in the vectorized form of $\overline{A}$. (output)

Return `numnz` (`int`) – Number of nonzero elements in $\overline{A}$.

Groups *Problem data - semidefinite*, *Inspecting the task*

Task.getbarcblocktriplet

```
def getbarcblocktriplet (subj, subk, subl, valjkl) -> num
```

Obtains $\overline{C}$ in block triplet form.

Parameters

- `subj` (`int[]`) – Symmetric matrix variable index. (output)
- `subk` (`int[]`) – Block row index. (output)
- `subl` (`int[]`) – Block column index. (output)
- `valjkl` (`float[]`) – The numerical value associated with each block triplet. (output)

Return `num` (`int`) – Number of elements in the block triplet form.

Groups *Problem data - semidefinite*, *Inspecting the task*

Task.getbarcidx

```
def getbarcidx (idx, sub, weights) -> j, num
```

Obtains information about an element in $\overline{C}$.

Parameters

- `idx` (`int`) – Index of the element for which information should be obtained. (input)
- `sub` (`int[]`) – Elements appearing the weighted sum. (output)
- `weights` (`float[]`) – Weights of terms in the weighted sum. (output)

Return

- j (int) – Row index in $\overline{C}$.
- num (int) – Number of terms in the weighted sum.

Groups *Problem data - semidefinite*, *Inspecting the task*

Task.getbarcidxinfo

```
def getbarcidxinfo (idx) -> num
```

Obtains the number of terms in the weighted sum that forms a particular element in $\overline{C}$.

Parameters idx (int) – Index of the element for which information should be obtained. The value is an index of a symmetric sparse variable. (input)

Return num (int) – Number of terms that appear in the weighted sum that forms the requested element.

Groups *Problem data - semidefinite*, *Inspecting the task*

Task.getbarcidxj

```
def getbarcidxj (idx) -> j
```

Obtains the row index of an element in $\overline{C}$.

Parameters idx (int) – Index of the element for which information should be obtained. (input)

Return j (int) – Row index in $\overline{C}$.

Groups *Problem data - semidefinite*, *Inspecting the task*

Task.getbarcsparsity

```
def getbarcsparsity (idxj) -> numnz
```

Internally only the nonzero elements of $\overline{C}$ are stored in a vector. This function is used to obtain the nonzero elements of $\overline{C}$ and their indexes in the internal vector representation (in idx). From the index detailed information about each nonzero $\overline{C}_j$ can be obtained using *Task.getbarcidxinfo* and *Task.getbarcidx*.

Parameters idxj (int[]) – Internal positions of the nonzeros elements in $\overline{C}$. (output)

Return numnz (int) – Number of nonzero elements in $\overline{C}$.

Groups *Problem data - semidefinite*, *Inspecting the task*

Task.getbarsj

```
def getbarsj (whichsol, j, barsj)
```

Obtains the dual solution for a semidefinite variable. Only the lower triangular part of $\overline{S}_j$ is returned because the matrix by construction is symmetric. The format is that the columns are stored sequentially in the natural order.

Parameters

- whichsol (*mosek.soltype*) – Selects a solution. (input)
- j (int) – Index of the semidefinite variable. (input)
- barsj (float[]) – Value of $\overline{S}_j$. (output)

Groups *Solution - semidefinite*

Task.getbarsslice

```
def getbarsslice (whichsol, first, last, slicesize, barsslice)
```

Obtains the dual solution for a sequence of semidefinite variables. The format is that matrices are stored sequentially, and in each matrix the columns are stored as in *Task.getbarsj*.

Parameters
- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (`int`) – Index of the first semidefinite variable in the slice. (input)
- `last` (`int`) – Index of the last semidefinite variable in the slice plus one. (input)
- `slicesize` (`int`) – Denotes the length of the array `barsslice`. (input)
- `barsslice` (`float[]`) – Dual solution values of symmetric matrix variables in the slice, stored sequentially. (output)

Groups *Solution - semidefinite*

Task.getbarvarname

```
def getbarvarname (i) -> name
```

Obtains the name of a semidefinite variable.

Parameters `i` (`int`) – Index of the variable. (input)
Return `name` (`str`) – The requested name is copied to this buffer.
Groups *Names*, *Inspecting the task*

Task.getbarvarnameindex

```
def getbarvarnameindex (somename) -> asgn, index
```

Obtains the index of semidefinite variable from its name.

Parameters `somename` (`str`) – The name of the variable. (input)
Return
- `asgn` (`int`) – Non-zero if the name `somename` is assigned to some semidefinite variable.
- `index` (`int`) – The index of a semidefinite variable with the name `somename` (if one exists).

Groups *Names*, *Inspecting the task*

Task.getbarvarnamelen

```
def getbarvarnamelen (i) -> len
```

Obtains the length of the name of a semidefinite variable.

Parameters `i` (`int`) – Index of the variable. (input)
Return `len` (`int`) – Returns the length of the indicated name.
Groups *Names*, *Inspecting the task*

Task.getbarxj

```
def getbarxj (whichsol, j, barxj)
```

Obtains the primal solution for a semidefinite variable. Only the lower triangular part of $\overline{X}_j$ is returned because the matrix by construction is symmetric. The format is that the columns are stored sequentially in the natural order.

Parameters

- whichsol (*mosek.soltype*) – Selects a solution. (input)
- j (int) – Index of the semidefinite variable. (input)
- barxj (float[]) – Value of $\overline{X}_j$. (output)

Groups *Solution - semidefinite*

Task.getbarxslice

```
def getbarxslice (whichsol, first, last, slicesize, barxslice)
```

Obtains the primal solution for a sequence of semidefinite variables. The format is that matrices are stored sequentially, and in each matrix the columns are stored as in *Task.getbarxj*.

Parameters

- whichsol (*mosek.soltype*) – Selects a solution. (input)
- first (int) – Index of the first semidefinite variable in the slice. (input)
- last (int) – Index of the last semidefinite variable in the slice plus one. (input)
- slicesize (int) – Denotes the length of the array barxslice. (input)
- barxslice (float[]) – Solution values of symmetric matrix variables in the slice, stored sequentially. (output)

Groups *Solution - semidefinite*

Task.getc

```
def getc (c)
```

Obtains all objective coefficients c.

Parameters c (float[]) – Linear terms of the objective as a dense vector. The length is the number of variables. (output)

Groups *Problem data - linear part*, *Inspecting the task*, *Problem data - variables*

Task.getcfix

```
def getcfix () -> cfix
```

Obtains the fixed term in the objective.

Return cfix (float) – Fixed term in the objective.

Groups *Problem data - linear part*, *Inspecting the task*

Task.getcj

```
def getcj (j) -> cj
```

Obtains one coefficient of c.

Parameters j (int) – Index of the variable for which the c coefficient should be obtained. (input)

Return cj (float) – The value of c_j.

Groups *Problem data - linear part*, *Inspecting the task*, *Problem data - variables*

Task.getclist

```
def getclist (subj, c)
```

Obtains a sequence of elements in c.

Parameters

- `subj` (`int[]`) – A list of variable indexes. (input)
- `c` (`float[]`) – Linear terms of the requested list of the objective as a dense vector. (output)

Groups *Inspecting the task*, *Problem data - linear part*

`Task.getconbound`

```
def getconbound (i) -> bk, bl, bu
```

Obtains bound information for one constraint.

Parameters `i` (`int`) – Index of the constraint for which the bound information should be obtained. (input)

Return

- `bk` (*`mosek.boundkey`*) – Bound keys.
- `bl` (`float`) – Values for lower bounds.
- `bu` (`float`) – Values for upper bounds.

Groups *Problem data - linear part*, *Inspecting the task*, *Problem data - bounds*, *Problem data - constraints*

`Task.getconboundslice`

```
def getconboundslice (first, last, bk, bl, bu)
```

Obtains bounds information for a slice of the constraints.

Parameters

- `first` (`int`) – First index in the sequence. (input)
- `last` (`int`) – Last index plus 1 in the sequence. (input)
- `bk` (*`mosek.boundkey`*`[]`) – Bound keys. (output)
- `bl` (`float[]`) – Values for lower bounds. (output)
- `bu` (`float[]`) – Values for upper bounds. (output)

Groups *Problem data - linear part*, *Inspecting the task*, *Problem data - bounds*, *Problem data - constraints*

`Task.getcone`

```
def getcone (k, submem) -> ct, conepar, nummem
```

Obtains a cone.

Parameters

- `k` (`int`) – Index of the cone. (input)
- `submem` (`int[]`) – Variable subscripts of the members in the cone. (output)

Return

- `ct` (*`mosek.conetype`*) – Specifies the type of the cone.
- `conepar` (`float`) – For the power cone it denotes the exponent alpha. For other cone types it is unused and can be set to 0.
- `nummem` (`int`) – Number of member variables in the cone.

Groups *Inspecting the task*, *Problem data - cones*

`Task.getconeinfo`

```
def getconeinfo (k) -> ct, conepar, nummem
```

Obtains information about a cone.

Parameters `k` (`int`) – Index of the cone. (input)

Return

- `ct` (*mosek.conetype*) – Specifies the type of the cone.
- `conepar` (`float`) – For the power cone it denotes the exponent alpha. For other cone types it is unused and can be set to 0.
- `nummem` (`int`) – Number of member variables in the cone.

Groups *Inspecting the task*, *Problem data - cones*

`Task.getconename`

```
def getconename (i) -> name
```

Obtains the name of a cone.

Parameters `i` (`int`) – Index of the cone. (input)

Return `name` (`str`) – The required name.

Groups *Names*, *Problem data - cones*, *Inspecting the task*

`Task.getconenameindex`

```
def getconenameindex (somename) -> asgn, index
```

Checks whether the name `somename` has been assigned to any cone. If it has been assigned to a cone, then the index of the cone is reported.

Parameters `somename` (`str`) – The name which should be checked. (input)

Return

- `asgn` (`int`) – Is non-zero if the name `somename` is assigned to some cone.
- `index` (`int`) – If the name `somename` is assigned to some cone, then `index` is the index of the cone.

Groups *Names*, *Problem data - cones*, *Inspecting the task*

`Task.getconenamelen`

```
def getconenamelen (i) -> len
```

Obtains the length of the name of a cone.

Parameters `i` (`int`) – Index of the cone. (input)

Return `len` (`int`) – Returns the length of the indicated name.

Groups *Names*, *Problem data - cones*, *Inspecting the task*

`Task.getconname`

```
def getconname (i) -> name
```

Obtains the name of a constraint.

Parameters `i` (`int`) – Index of the constraint. (input)

Return `name` (`str`) – The required name.

Groups *Names*, *Problem data - linear part*, *Problem data - constraints*, *Inspecting the task*

Task.getconnameindex

```
def getconnameindex (somename) -> asgn, index
```

Checks whether the name somename has been assigned to any constraint. If so, the index of the constraint is reported.

Parameters somename (str) – The name which should be checked. (input)

Return

- asgn (int) – Is non-zero if the name somename is assigned to some constraint.
- index (int) – If the name somename is assigned to a constraint, then index is the index of the constraint.

Groups *Names*, *Problem data - linear part*, *Problem data - constraints*, *Inspecting the task*

Task.getconnamelen

```
def getconnamelen (i) -> len
```

Obtains the length of the name of a constraint.

Parameters i (int) – Index of the constraint. (input)

Return len (int) – Returns the length of the indicated name.

Groups *Names*, *Problem data - linear part*, *Problem data - constraints*, *Inspecting the task*

Task.getcslice

```
def getcslice (first, last, c)
```

Obtains a sequence of elements in c.

Parameters

- first (int) – First index in the sequence. (input)
- last (int) – Last index plus 1 in the sequence. (input)
- c (float[]) – Linear terms of the requested slice of the objective as a dense vector. The length is last-first. (output)

Groups *Inspecting the task*, *Problem data - linear part*

Task.getdimbarvarj

```
def getdimbarvarj (j) -> dimbarvarj
```

Obtains the dimension of a symmetric matrix variable.

Parameters j (int) – Index of the semidefinite variable whose dimension is requested. (input)

Return dimbarvarj (int) – The dimension of the j-th semidefinite variable.

Groups *Inspecting the task*, *Problem data - semidefinite*

Task.getdouinf

```
def getdouinf (whichdinf) -> dvalue
```

Obtains a double information item from the task information database.

Parameters whichdinf (*mosek.dinfitem*) – Specifies a double information item. (input)

Return dvalue (float) – The value of the required double information item.

Groups *Information items and statistics*

Task.getdouparam

```
def getdouparam (param) -> parvalue
```

Obtains the value of a double parameter.

Parameters param (*mosek.dparam*) – Which parameter. (input)

Return parvalue (float) – Parameter value.

Groups *Parameters*

Task.getdualobj

```
def getdualobj (whichsol) -> dualobj
```

Computes the dual objective value associated with the solution. Note that if the solution is a primal infeasibility certificate, then the fixed term in the objective value is not included.

Moreover, since there is no dual solution associated with an integer solution, an error will be reported if the dual objective value is requested for the integer solution.

Parameters whichsol (*mosek.soltype*) – Selects a solution. (input)

Return dualobj (float) – Objective value corresponding to the dual solution.

Groups *Solution information*, *Solution - dual*

Task.getdualsolutionnorms

```
def getdualsolutionnorms (whichsol) -> nrmy, nrmslc, nrmsuc, nrmslx, nrmsux, nrmsnx,␣
↪nrmbars
```

Compute norms of the dual solution.

Parameters whichsol (*mosek.soltype*) – Selects a solution. (input)

Return

- nrmy (float) – The norm of the y vector.
- nrmslc (float) – The norm of the s_l^c vector.
- nrmsuc (float) – The norm of the s_u^c vector.
- nrmslx (float) – The norm of the s_l^x vector.
- nrmsux (float) – The norm of the s_u^x vector.
- nrmsnx (float) – The norm of the s_n^x vector.
- nrmbars (float) – The norm of the $\overline{S}$ vector.

Groups *Solution information*

Task.getdviolbarvar

```
def getdviolbarvar (whichsol, sub, viol)
```

Let $(\overline{S}_j)^*$ be the value of variable $\overline{S}_j$ for the specified solution. Then the dual violation of the solution associated with variable $\overline{S}_j$ is given by

$$\max(-\lambda_{\min}(\overline{S}_j),\ 0.0).$$

Both when the solution is a certificate of primal infeasibility and when it is dual feasible solution the violation should be small.

Parameters

- whichsol (*mosek.soltype*) – Selects a solution. (input)
- sub (int[]) – An array of indexes of $\overline{X}$ variables. (input)
- viol (float[]) – viol[k] is the violation of the solution for the constraint $\overline{S}_{\texttt{sub}[k]} \in \mathcal{S}_+$. (output)

Groups *Solution information*

Task.getdviolcon

```
def getdviolcon (whichsol, sub, viol)
```

The violation of the dual solution associated with the i-th constraint is computed as follows

$$\max(\rho((s_l^c)_i^*, (b_l^c)_i),\ \rho((s_u^c)_i^*, -(b_u^c)_i),\ |-y_i + (s_l^c)_i^* - (s_u^c)_i^*|)$$

where

$$\rho(x,l) = \begin{cases} -x, & l > -\infty, \\ |x|, & \text{otherwise.} \end{cases}$$

Both when the solution is a certificate of primal infeasibility or it is a dual feasible solution the violation should be small.

Parameters

- whichsol (*mosek.soltype*) – Selects a solution. (input)
- sub (int[]) – An array of indexes of constraints. (input)
- viol (float[]) – viol[k] is the violation of dual solution associated with the constraint sub[k]. (output)

Groups *Solution information*

Task.getdviolcones

```
def getdviolcones (whichsol, sub, viol)
```

Let $(s_n^x)^*$ be the value of variable (s_n^x) for the specified solution. For simplicity let us assume that s_n^x is a member of a quadratic cone, then the violation is computed as follows

$$\begin{cases} \max(0, (\|s_n^x\|_{2:n}^* - (s_n^x)_1^*)/\sqrt{2}, & (s_n^x)^* \geq -\|(s_n^x)_{2:n}^*\|, \\ \|(s_n^x)^*\|, & \text{otherwise.} \end{cases}$$

Both when the solution is a certificate of primal infeasibility or when it is a dual feasible solution the violation should be small.

Parameters

- whichsol (*mosek.soltype*) – Selects a solution. (input)
- sub (int[]) – An array of indexes of conic constraints. (input)
- viol (float[]) – viol[k] is the violation of the dual solution associated with the conic constraint sub[k]. (output)

Groups *Solution information*

Task.getdviolvar

```
def getdviolvar (whichsol, sub, viol)
```

The violation of the dual solution associated with the j-th variable is computed as follows

$$\max\left(\rho((s_l^x)_j^*, (b_l^x)_j),\ \rho((s_u^x)_j^*, -(b_u^x)_j),\ |\sum_{i=0}^{numcon-1} a_{ij} y_i + (s_l^x)_j^* - (s_u^x)_j^* - \tau c_j|\right)$$

where

$$\rho(x,l) = \begin{cases} -x, & l > -\infty, \\ |x|, & \text{otherwise} \end{cases}$$

and $\tau = 0$ if the solution is a certificate of primal infeasibility and $\tau = 1$ otherwise. The formula for computing the violation is only shown for the linear case but is generalized appropriately for the more general problems. Both when the solution is a certificate of primal infeasibility or when it is a dual feasible solution the violation should be small.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `sub` (`int[]`) – An array of indexes of x variables. (input)
- `viol` (`float[]`) – `viol[k]` is the violation of dual solution associated with the variable `sub[k]`. (output)

Groups *Solution information*

`Task.getinfeasiblesubproblem`

```
def getinfeasiblesubproblem (whichsol) -> inftask
```

Given the solution is a certificate of primal or dual infeasibility then a primal or dual infeasible subproblem is obtained respectively. The subproblem tends to be much smaller than the original problem and hence it is easier to locate the infeasibility inspecting the subproblem than the original problem.

For the procedure to be useful it is important to assign meaningful names to constraints, variables etc. in the original task because those names will be duplicated in the subproblem.

The function is only applicable to linear and conic quadratic optimization problems.

For more information see Sec. 8.3 and Sec. 14.2.

Parameters `whichsol` (*mosek.soltype*) – Which solution to use when determining the infeasible subproblem. (input)

Return `inftask` (*Task*) – A new task containing the infeasible subproblem.

Groups *Infeasibility diagnostic*

`Task.getintinf`

```
def getintinf (whichiinf) -> ivalue
```

Obtains an integer information item from the task information database.

Parameters `whichiinf` (*mosek.iinfitem*) – Specifies an integer information item. (input)

Return `ivalue` (`int`) – The value of the required integer information item.

Groups *Information items and statistics*

`Task.getintparam`

```
def getintparam (param) -> parvalue
```

Obtains the value of an integer parameter.

Parameters `param` (*mosek.iparam*) – Which parameter. (input)

Return `parvalue` (`int`) – Parameter value.

Groups *Parameters*

`Task.getlenbarvarj`

```
def getlenbarvarj (j) -> lenbarvarj
```

Obtains the length of the j-th semidefinite variable i.e. the number of elements in the lower triangular part.

Parameters `j` (`int`) – Index of the semidefinite variable whose length if requested. (input)

Return `lenbarvarj` (`int`) – Number of scalar elements in the lower triangular part of the semidefinite variable.

Groups *Inspecting the task*, *Problem data - semidefinite*

`Task.getlintinf`

```
def getlintinf (whichliinf) -> ivalue
```

Obtains a long integer information item from the task information database.

Parameters `whichliinf` (*mosek.liinfitem*) – Specifies a long information item. (input)

Return `ivalue` (`int`) – The value of the required long integer information item.

Groups *Information items and statistics*

`Task.getmaxnumanz`

```
def getmaxnumanz () -> maxnumanz
```

Obtains number of preallocated non-zeros in A. When this number of non-zeros is reached **MOSEK** will automatically allocate more space for A.

Return `maxnumanz` (`int`) – Number of preallocated non-zero linear matrix elements.

Groups *Inspecting the task*, *Problem data - linear part*

`Task.getmaxnumbarvar`

```
def getmaxnumbarvar () -> maxnumbarvar
```

Obtains maximum number of symmetric matrix variables for which space is currently preallocated.

Return `maxnumbarvar` (`int`) – Maximum number of symmetric matrix variables for which space is currently preallocated.

Groups *Inspecting the task*, *Problem data - semidefinite*

`Task.getmaxnumcon`

```
def getmaxnumcon () -> maxnumcon
```

Obtains the number of preallocated constraints in the optimization task. When this number of constraints is reached **MOSEK** will automatically allocate more space for constraints.

Return `maxnumcon` (`int`) – Number of preallocated constraints in the optimization task.

Groups *Inspecting the task*, *Problem data - linear part*, *Problem data - constraints*

`Task.getmaxnumcone`

```
def getmaxnumcone () -> maxnumcone
```

Obtains the number of preallocated cones in the optimization task. When this number of cones is reached **MOSEK** will automatically allocate space for more cones.

Return `maxnumcone` (`int`) – Number of preallocated conic constraints in the optimization task.

Groups *Inspecting the task*, *Problem data - cones*

`Task.getmaxnumqnz`

```
def getmaxnumqnz () -> maxnumqnz
```

Obtains the number of preallocated non-zeros for Q (both objective and constraints). When this number of non-zeros is reached **MOSEK** will automatically allocate more space for Q.

Return `maxnumqnz` (`int`) – Number of non-zero elements preallocated in quadratic coefficient matrices.

Groups *Inspecting the task*, *Problem data - quadratic part*

`Task.getmaxnumvar`

```
def getmaxnumvar () -> maxnumvar
```

Obtains the number of preallocated variables in the optimization task. When this number of variables is reached **MOSEK** will automatically allocate more space for variables.

Return `maxnumvar` (`int`) – Number of preallocated variables in the optimization task.

Groups *Inspecting the task*, *Problem data - linear part*, *Problem data - variables*

`Task.getmemusage`

```
def getmemusage () -> meminuse, maxmemuse
```

Obtains information about the amount of memory used by a task.

Return

- `meminuse` (`int`) – Amount of memory currently used by the `task`.
- `maxmemuse` (`int`) – Maximum amount of memory used by the `task` until now.

Groups *System, memory and debugging*

`Task.getnumanz`

```
def getnumanz () -> numanz
```

Obtains the number of non-zeros in A.

Return `numanz` (`int`) – Number of non-zero elements in the linear constraint matrix.

Groups *Inspecting the task*, *Problem data - linear part*

`Task.getnumanz64`

```
def getnumanz64 () -> numanz
```

Obtains the number of non-zeros in A.

Return `numanz` (`int`) – Number of non-zero elements in the linear constraint matrix.

Groups *Inspecting the task*, *Problem data - linear part*

`Task.getnumbarablocktriplets`

```
def getnumbarablocktriplets () -> num
```

Obtains an upper bound on the number of elements in the block triplet form of $\overline{A}$.

Return `num` (`int`) – An upper bound on the number of elements in the block triplet form of $\overline{A}$.

Groups *Problem data - semidefinite*, *Inspecting the task*

`Task.getnumbaranz`

```
def getnumbaranz () -> nz
```

Get the number of nonzero elements in $\overline{A}$.

Return `nz` (`int`) – The number of nonzero block elements in $\overline{A}$ i.e. the number of $\overline{A}_{ij}$ elements that are nonzero.

Groups *Problem data - semidefinite*, *Inspecting the task*

`Task.getnumbarcblocktriplets`

```
def getnumbarcblocktriplets () -> num
```

Obtains an upper bound on the number of elements in the block triplet form of $\overline{C}$.

Return `num` (`int`) – An upper bound on the number of elements in the block triplet form of $\overline{C}$.

Groups *Problem data - semidefinite*, *Inspecting the task*

`Task.getnumbarcnz`

```
def getnumbarcnz () -> nz
```

Obtains the number of nonzero elements in $\overline{C}$.

Return `nz` (`int`) – The number of nonzeros in $\overline{C}$ i.e. the number of elements $\overline{C}_j$ that are nonzero.

Groups *Problem data - semidefinite*, *Inspecting the task*

`Task.getnumbarvar`

```
def getnumbarvar () -> numbarvar
```

Obtains the number of semidefinite variables.

Return `numbarvar` (`int`) – Number of semidefinite variables in the problem.

Groups *Inspecting the task*, *Problem data - semidefinite*

`Task.getnumcon`

```
def getnumcon () -> numcon
```

Obtains the number of constraints.

Return `numcon` (`int`) – Number of constraints.

Groups *Problem data - linear part*, *Problem data - constraints*, *Inspecting the task*

`Task.getnumcone`

```
def getnumcone () -> numcone
```

Obtains the number of cones.

Return numcone (int) – Number of conic constraints.

Groups *Problem data - cones*, *Inspecting the task*

Task.getnumconemem

```
def getnumconemem (k) -> nummem
```

Obtains the number of members in a cone.

Parameters k (int) – Index of the cone. (input)

Return nummem (int) – Number of member variables in the cone.

Groups *Problem data - cones*, *Inspecting the task*

Task.getnumintvar

```
def getnumintvar () -> numintvar
```

Obtains the number of integer-constrained variables.

Return numintvar (int) – Number of integer variables.

Groups *Inspecting the task*, *Problem data - variables*

Task.getnumparam

```
def getnumparam (partype) -> numparam
```

Obtains the number of parameters of a given type.

Parameters partype (*mosek.parametertype*) – Parameter type. (input)

Return numparam (int) – The number of parameters of type partype.

Groups *Inspecting the task*, *Parameters*

Task.getnumqconknz

```
def getnumqconknz (k) -> numqcnz
```

Obtains the number of non-zero quadratic terms in a constraint.

Parameters k (int) – Index of the constraint for which the number quadratic terms should be obtained. (input)

Return numqcnz (int) – Number of quadratic terms.

Groups *Inspecting the task*, *Problem data - constraints*, *Problem data - quadratic part*

Task.getnumqobjnz

```
def getnumqobjnz () -> numqonz
```

Obtains the number of non-zero quadratic terms in the objective.

Return numqonz (int) – Number of non-zero elements in the quadratic objective terms.

Groups *Inspecting the task*, *Problem data - quadratic part*

Task.getnumsymmat

```
def getnumsymmat () -> num
```

Obtains the number of symmetric matrices stored in the vector E.

Return `num` (`int`) – The number of symmetric sparse matrices.
Groups *Problem data - semidefinite*, *Inspecting the task*

`Task.getnumvar`

```
def getnumvar () -> numvar
```

Obtains the number of variables.

Return `numvar` (`int`) – Number of variables.
Groups *Inspecting the task*, *Problem data - variables*

`Task.getobjname`

```
def getobjname () -> objname
```

Obtains the name assigned to the objective function.

Return `objname` (`str`) – Assigned the objective name.
Groups *Inspecting the task*, *Names*

`Task.getobjnamelen`

```
def getobjnamelen () -> len
```

Obtains the length of the name assigned to the objective function.

Return `len` (`int`) – Assigned the length of the objective name.
Groups *Inspecting the task*, *Names*

`Task.getobjsense`

```
def getobjsense () -> sense
```

Gets the objective sense of the task.

Return `sense` (*`mosek.objsense`*) – The returned objective sense.
Groups *Problem data - linear part*

`Task.getprimalobj`

```
def getprimalobj (whichsol) -> primalobj
```

Computes the primal objective value for the desired solution. Note that if the solution is an infeasibility certificate, then the fixed term in the objective is not included.

Parameters `whichsol` (*`mosek.soltype`*) – Selects a solution. (input)
Return `primalobj` (`float`) – Objective value corresponding to the primal solution.
Groups *Solution information*, *Solution - primal*

`Task.getprimalsolutionnorms`

```
def getprimalsolutionnorms (whichsol) -> nrmxc, nrmxx, nrmbarx
```

Compute norms of the primal solution.

Parameters whichsol (*mosek.soltype*) – Selects a solution. (input)

Return

- nrmxc (float) – The norm of the x^c vector.
- nrmxx (float) – The norm of the x vector.
- nrmbarx (float) – The norm of the $\overline{X}$ vector.

Groups *Solution information*

Task.getprobtype

```
def getprobtype () -> probtype
```

Obtains the problem type.

Return probtype (*mosek.problemtype*) – The problem type.

Groups *Inspecting the task*

Task.getprosta

```
def getprosta (whichsol) -> prosta
```

Obtains the problem status.

Parameters whichsol (*mosek.soltype*) – Selects a solution. (input)

Return prosta (*mosek.prosta*) – Problem status.

Groups *Solution information*

Task.getpviolbarvar

```
def getpviolbarvar (whichsol, sub, viol)
```

Computes the primal solution violation for a set of semidefinite variables. Let $(\overline{X}_j)^*$ be the value of the variable $\overline{X}_j$ for the specified solution. Then the primal violation of the solution associated with variable $\overline{X}_j$ is given by

$$\max(-\lambda_{\min}(\overline{X}_j),\ 0.0).$$

Both when the solution is a certificate of dual infeasibility or when it is primal feasible the violation should be small.

Parameters

- whichsol (*mosek.soltype*) – Selects a solution. (input)
- sub (int[]) – An array of indexes of $\overline{X}$ variables. (input)
- viol (float[]) – viol[k] is how much the solution violates the constraint $\overline{X}_{\mathtt{sub}[k]} \in \mathcal{S}_+$. (output)

Groups *Solution information*

Task.getpviolcon

```
def getpviolcon (whichsol, sub, viol)
```

Computes the primal solution violation for a set of constraints. The primal violation of the solution associated with the i-th constraint is given by

$$\max(\tau l_i^c - (x_i^c)^*,\ (x_i^c)^* - \tau u_i^c),\ |\sum_{j=0}^{numvar-1} a_{ij} x_j^* - x_i^c|)$$

where $\tau = 0$ if the solution is a certificate of dual infeasibility and $\tau = 1$ otherwise. Both when the solution is a certificate of dual infeasibility and when it is primal feasible the violation should be small. The above formula applies for the linear case but is appropriately generalized in other cases.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `sub` (`int[]`) – An array of indexes of constraints. (input)
- `viol` (`float[]`) – `viol[k]` is the violation associated with the solution for the constraint `sub[k]`. (output)

Groups *Solution information*

`Task.getpviolcones`

```
def getpviolcones (whichsol, sub, viol)
```

Computes the primal solution violation for a set of conic constraints. Let x^* be the value of the variable x for the specified solution. For simplicity let us assume that x is a member of a quadratic cone, then the violation is computed as follows

$$\begin{cases} \max(0, \|x_{2:n}\| - x_1)/\sqrt{2}, & x_1 \geq -\|x_{2:n}\|, \\ \|x\|, & \text{otherwise.} \end{cases}$$

Both when the solution is a certificate of dual infeasibility or when it is primal feasible the violation should be small.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `sub` (`int[]`) – An array of indexes of conic constraints. (input)
- `viol` (`float[]`) – `viol[k]` is the violation of the solution associated with the conic constraint number `sub[k]`. (output)

Groups *Solution information*

`Task.getpviolvar`

```
def getpviolvar (whichsol, sub, viol)
```

Computes the primal solution violation associated to a set of variables. Let x_j^* be the value of x_j for the specified solution. Then the primal violation of the solution associated with variable x_j is given by

$$\max(\tau l_j^x - x_j^*,\ x_j^* - \tau u_j^x,\ 0).$$

where $\tau = 0$ if the solution is a certificate of dual infeasibility and $\tau = 1$ otherwise. Both when the solution is a certificate of dual infeasibility and when it is primal feasible the violation should be small.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `sub` (`int[]`) – An array of indexes of x variables. (input)
- `viol` (`float[]`) – `viol[k]` is the violation associated with the solution for the variable $x_{\mathtt{sub[k]}}$. (output)

Groups *Solution information*

`Task.getqconk`

```
def getqconk (k, qcsubi, qcsubj, qcval) -> numqcnz
```

Obtains all the quadratic terms in a constraint. The quadratic terms are stored sequentially in `qcsubi`, `qcsubj`, and `qcval`.

Parameters
- `k` (`int`) – Which constraint. (input)
- `qcsubi` (`int[]`) – Row subscripts for quadratic constraint matrix. (output)
- `qcsubj` (`int[]`) – Column subscripts for quadratic constraint matrix. (output)
- `qcval` (`float[]`) – Quadratic constraint coefficient values. (output)

Return `numqcnz` (`int`) – Number of quadratic terms.

Groups *Inspecting the task*, *Problem data - quadratic part*, *Problem data - constraints*

`Task.getqobj`

```
def getqobj (qosubi, qosubj, qoval) -> numqonz
```

Obtains the quadratic terms in the objective. The required quadratic terms are stored sequentially in `qosubi`, `qosubj`, and `qoval`.

Parameters
- `qosubi` (`int[]`) – Row subscripts for quadratic objective coefficients. (output)
- `qosubj` (`int[]`) – Column subscripts for quadratic objective coefficients. (output)
- `qoval` (`float[]`) – Quadratic objective coefficient values. (output)

Return `numqonz` (`int`) – Number of non-zero elements in the quadratic objective terms.

Groups *Inspecting the task*, *Problem data - quadratic part*

`Task.getqobjij`

```
def getqobjij (i, j) -> qoij
```

Obtains one coefficient q_{ij}^o in the quadratic term of the objective.

Parameters
- `i` (`int`) – Row index of the coefficient. (input)
- `j` (`int`) – Column index of coefficient. (input)

Return `qoij` (`float`) – The required coefficient.

Groups *Inspecting the task*, *Problem data - quadratic part*

`Task.getreducedcosts`

```
def getreducedcosts (whichsol, first, last, redcosts)
```

Computes the reduced costs for a slice of variables and returns them in the array `redcosts` i.e.

$$\texttt{redcosts}[j-\texttt{first}] = (s_l^x)_j - (s_u^x)_j,\ \ j = \texttt{first}, \ldots, \texttt{last}-1 \tag{15.2}$$

Parameters
- `whichsol` (*`mosek.soltype`*) – Selects a solution. (input)
- `first` (`int`) – The index of the first variable in the sequence. (input)
- `last` (`int`) – The index of the last variable in the sequence plus 1. (input)

- `redcosts` (`float[]`) – The reduced costs for the required slice of variables. (output)

Groups *Solution - dual*

`Task.getskc`

```
def getskc (whichsol, skc)
```

Obtains the status keys for the constraints.

Parameters
- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `skc` (*mosek.stakey*`[]`) – Status keys for the constraints. (output)

Groups *Solution information*

`Task.getskcslice`

```
def getskcslice (whichsol, first, last, skc)
```

Obtains the status keys for a slice of the constraints.

Parameters
- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (`int`) – First index in the sequence. (input)
- `last` (`int`) – Last index plus 1 in the sequence. (input)
- `skc` (*mosek.stakey*`[]`) – Status keys for the constraints. (output)

Groups *Solution information*

`Task.getskn`

```
def getskn (whichsol, skn)
```

Obtains the status keys for the conic constraints.

Parameters
- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `skn` (*mosek.stakey*`[]`) – Status keys for the conic constraints. (output)

Groups *Solution information*

`Task.getskx`

```
def getskx (whichsol, skx)
```

Obtains the status keys for the scalar variables.

Parameters
- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `skx` (*mosek.stakey*`[]`) – Status keys for the variables. (output)

Groups *Solution information*

`Task.getskxslice`

```
def getskxslice (whichsol, first, last, skx)
```

Obtains the status keys for a slice of the scalar variables.

Parameters

- whichsol (*mosek.soltype*) – Selects a solution. (input)
- first (int) – First index in the sequence. (input)
- last (int) – Last index plus 1 in the sequence. (input)
- skx (*mosek.stakey* []) – Status keys for the variables. (output)

Groups *Solution information*

Task.getslc

```
def getslc (whichsol, slc)
```

Obtains the s_l^c vector for a solution.

Parameters

- whichsol (*mosek.soltype*) – Selects a solution. (input)
- slc (float[]) – Dual variables corresponding to the lower bounds on the constraints. (output)

Groups *Solution - dual*

Task.getslcslice

```
def getslcslice (whichsol, first, last, slc)
```

Obtains a slice of the s_l^c vector for a solution.

Parameters

- whichsol (*mosek.soltype*) – Selects a solution. (input)
- first (int) – First index in the sequence. (input)
- last (int) – Last index plus 1 in the sequence. (input)
- slc (float[]) – Dual variables corresponding to the lower bounds on the constraints. (output)

Groups *Solution - dual*

Task.getslx

```
def getslx (whichsol, slx)
```

Obtains the s_l^x vector for a solution.

Parameters

- whichsol (*mosek.soltype*) – Selects a solution. (input)
- slx (float[]) – Dual variables corresponding to the lower bounds on the variables. (output)

Groups *Solution - dual*

Task.getslxslice

```
def getslxslice (whichsol, first, last, slx)
```

Obtains a slice of the s_l^x vector for a solution.

Parameters

- whichsol (*mosek.soltype*) – Selects a solution. (input)
- first (int) – First index in the sequence. (input)
- last (int) – Last index plus 1 in the sequence. (input)
- slx (float[]) – Dual variables corresponding to the lower bounds on the variables. (output)

Groups *Solution - dual*

`Task.getsnx`

```
def getsnx (whichsol, snx)
```

Obtains the s_n^x vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `snx` (`float[]`) – Dual variables corresponding to the conic constraints on the variables. (output)

Groups *Solution - dual*

`Task.getsnxslice`

```
def getsnxslice (whichsol, first, last, snx)
```

Obtains a slice of the s_n^x vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (`int`) – First index in the sequence. (input)
- `last` (`int`) – Last index plus 1 in the sequence. (input)
- `snx` (`float[]`) – Dual variables corresponding to the conic constraints on the variables. (output)

Groups *Solution - dual*

`Task.getsolsta`

```
def getsolsta (whichsol) -> solsta
```

Obtains the solution status.

Parameters `whichsol` (*mosek.soltype*) – Selects a solution. (input)
Return `solsta` (*mosek.solsta*) – Solution status.
Groups *Solution information*

`Task.getsolution`

```
def getsolution (whichsol, skc, skx, skn, xc, xx, y, slc, suc, slx, sux, snx) -> prosta,␣
↪solsta
```

Obtains the complete solution.

Consider the case of linear programming. The primal problem is given by

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

and the corresponding dual problem is

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

A conic optimization problem has the same primal variables as in the linear case. Recall that the dual of a conic optimization problem is given by:

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

The mapping between variables and arguments to the function is as follows:

- `xx` : Corresponds to variable x (also denoted x^x).
- `xc` : Corresponds to $x^c := Ax$.
- `y` : Corresponds to variable y.
- `slc`: Corresponds to variable s_l^c.
- `suc`: Corresponds to variable s_u^c.
- `slx`: Corresponds to variable s_l^x.
- `sux`: Corresponds to variable s_u^x.
- `snx`: Corresponds to variable s_n^x.

The meaning of the values returned by this function depend on the *solution status* returned in the argument `solsta`. The most important possible values of `solsta` are:

- *solsta.optimal* : An optimal solution satisfying the optimality criteria for continuous problems is returned.
- *solsta.integer_optimal* : An optimal solution satisfying the optimality criteria for integer problems is returned.
- *solsta.prim_feas* : A solution satisfying the feasibility criteria.
- *solsta.prim_infeas_cer* : A primal certificate of infeasibility is returned.
- *solsta.dual_infeas_cer* : A dual certificate of infeasibility is returned.

In order to retrieve the primal and dual values of semidefinite variables see *Task.getbarxj* and *Task.getbarsj*.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `skc` (*mosek.stakey* `[]`) – Status keys for the constraints. (output)
- `skx` (*mosek.stakey* `[]`) – Status keys for the variables. (output)
- `skn` (*mosek.stakey* `[]`) – Status keys for the conic constraints. (output)
- `xc` (`float[]`) – Primal constraint solution. (output)
- `xx` (`float[]`) – Primal variable solution. (output)
- `y` (`float[]`) – Vector of dual variables corresponding to the constraints. (output)
- `slc` (`float[]`) – Dual variables corresponding to the lower bounds on the constraints. (output)
- `suc` (`float[]`) – Dual variables corresponding to the upper bounds on the constraints. (output)
- `slx` (`float[]`) – Dual variables corresponding to the lower bounds on the variables. (output)
- `sux` (`float[]`) – Dual variables corresponding to the upper bounds on the variables. (output)
- `snx` (`float[]`) – Dual variables corresponding to the conic constraints on the variables. (output)

Return

- `prosta` (*mosek.prosta*) – Problem status.

- solsta (*mosek.solsta*) – Solution status.

Groups *Solution information*, *Solution - primal*, *Solution - dual*

Task.getsolutioninfo

```
def getsolutioninfo (whichsol) -> pobj, pviolcon, pviolvar, pviolbarvar, pviolcone,␣
↪pviolitg, dobj, dviolcon, dviolvar, dviolbarvar, dviolcone
```

Obtains information about a solution.

Parameters whichsol (*mosek.soltype*) – Selects a solution. (input)

Return

- pobj (float) – The primal objective value as computed by *Task.getprimalobj*.
- pviolcon (float) – Maximal primal violation of the solution associated with the x^c variables where the violations are computed by *Task.getpviolcon*.
- pviolvar (float) – Maximal primal violation of the solution for the x variables where the violations are computed by *Task.getpviolvar*.
- pviolbarvar (float) – Maximal primal violation of solution for the $\overline{X}$ variables where the violations are computed by *Task.getpviolbarvar*.
- pviolcone (float) – Maximal primal violation of solution for the conic constraints where the violations are computed by *Task.getpviolcones*.
- pviolitg (float) – Maximal violation in the integer constraints. The violation for an integer variable x_j is given by $\min(x_j - \lfloor x_j \rfloor, \lceil x_j \rceil - x_j)$. This number is always zero for the interior-point and basic solutions.
- dobj (float) – Dual objective value as computed by *Task.getdualobj*.
- dviolcon (float) – Maximal violation of the dual solution associated with the x^c variable as computed by *Task.getdviolcon*.
- dviolvar (float) – Maximal violation of the dual solution associated with the x variable as computed by *Task.getdviolvar*.
- dviolbarvar (float) – Maximal violation of the dual solution associated with the $\overline{S}$ variable as computed by *Task.getdviolbarvar*.
- dviolcone (float) – Maximal violation of the dual solution associated with the dual conic constraints as computed by *Task.getdviolcones*.

Groups *Solution information*

Task.getsolutionslice

```
def getsolutionslice (whichsol, solitem, first, last, values)
```

Obtains a slice of one item from the solution. The format of the solution is exactly as in *Task.getsolution*. The parameter solitem determines which of the solution vectors should be returned.

Parameters

- whichsol (*mosek.soltype*) – Selects a solution. (input)
- solitem (*mosek.solitem*) – Which part of the solution is required. (input)
- first (int) – First index in the sequence. (input)
- last (int) – Last index plus 1 in the sequence. (input)
- values (float[]) – The values in the required sequence are stored sequentially in values. (output)

Groups *Solution - primal*, *Solution - dual*, *Solution information*

Task.getsparsesymmat

```
def getsparsesymmat (idx, subi, subj, valij)
```

Get a single symmetric matrix from the matrix store.

Parameters

- `idx` (`int`) – Index of the matrix to retrieve. (input)
- `subi` (`int[]`) – Row subscripts of the matrix non-zero elements. (output)
- `subj` (`int[]`) – Column subscripts of the matrix non-zero elements. (output)
- `valij` (`float[]`) – Coefficients of the matrix non-zero elements. (output)

Groups *Problem data - semidefinite*, *Inspecting the task*

`Task.getstrparam`

```
def getstrparam (param) -> len, parvalue
```

Obtains the value of a string parameter.

Parameters `param` (*`mosek.sparam`*) – Which parameter. (input)

Return

- `len` (`int`) – The length of the parameter value.
- `parvalue` (`str`) – Parameter value.

Groups *Names*, *Parameters*

`Task.getstrparamlen`

```
def getstrparamlen (param) -> len
```

Obtains the length of a string parameter.

Parameters `param` (*`mosek.sparam`*) – Which parameter. (input)

Return `len` (`int`) – The length of the parameter value.

Groups *Names*, *Parameters*

`Task.getsuc`

```
def getsuc (whichsol, suc)
```

Obtains the s_u^c vector for a solution.

Parameters

- `whichsol` (*`mosek.soltype`*) – Selects a solution. (input)
- `suc` (`float[]`) – Dual variables corresponding to the upper bounds on the constraints. (output)

Groups *Solution - dual*

`Task.getsucslice`

```
def getsucslice (whichsol, first, last, suc)
```

Obtains a slice of the s_u^c vector for a solution.

Parameters

- `whichsol` (*`mosek.soltype`*) – Selects a solution. (input)
- `first` (`int`) – First index in the sequence. (input)
- `last` (`int`) – Last index plus 1 in the sequence. (input)
- `suc` (`float[]`) – Dual variables corresponding to the upper bounds on the constraints. (output)

Groups *Solution - dual*

Task.getsux

```
def getsux (whichsol, sux)
```

Obtains the s_u^x vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `sux` (`float[]`) – Dual variables corresponding to the upper bounds on the variables. (output)

Groups *Solution - dual*

Task.getsuxslice

```
def getsuxslice (whichsol, first, last, sux)
```

Obtains a slice of the s_u^x vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (`int`) – First index in the sequence. (input)
- `last` (`int`) – Last index plus 1 in the sequence. (input)
- `sux` (`float[]`) – Dual variables corresponding to the upper bounds on the variables. (output)

Groups *Solution - dual*

Task.getsymmatinfo

```
def getsymmatinfo (idx) -> dim, nz, type
```

MOSEK maintains a vector denoted by E of symmetric data matrices. This function makes it possible to obtain important information about a single matrix in E.

Parameters `idx` (`int`) – Index of the matrix for which information is requested. (input)

Return

- `dim` (`int`) – Returns the dimension of the requested matrix.
- `nz` (`int`) – Returns the number of non-zeros in the requested matrix.
- `type` (*mosek.symmattype*) – Returns the type of the requested matrix.

Groups *Problem data - semidefinite*, *Inspecting the task*

Task.gettaskname

```
def gettaskname () -> taskname
```

Obtains the name assigned to the task.

Return `taskname` (`str`) – Returns the task name.

Groups *Names*, *Inspecting the task*

Task.gettasknamelen

```
def gettasknamelen () -> len
```

Obtains the length the task name.

Return `len` (`int`) – Returns the length of the task name.

Groups *Names*, *Inspecting the task*

`Task.getvarbound`

```
def getvarbound (i) -> bk, bl, bu
```

Obtains bound information for one variable.

Parameters `i` (`int`) – Index of the variable for which the bound information should be obtained. (input)

Return

- `bk` (*mosek.boundkey*) – Bound keys.
- `bl` (`float`) – Values for lower bounds.
- `bu` (`float`) – Values for upper bounds.

Groups *Problem data - linear part*, *Inspecting the task*, *Problem data - bounds*, *Problem data - variables*

`Task.getvarboundslice`

```
def getvarboundslice (first, last, bk, bl, bu)
```

Obtains bounds information for a slice of the variables.

Parameters

- `first` (`int`) – First index in the sequence. (input)
- `last` (`int`) – Last index plus 1 in the sequence. (input)
- `bk` (*mosek.boundkey* `[]`) – Bound keys. (output)
- `bl` (`float[]`) – Values for lower bounds. (output)
- `bu` (`float[]`) – Values for upper bounds. (output)

Groups *Problem data - linear part*, *Inspecting the task*, *Problem data - bounds*, *Problem data - variables*

`Task.getvarname`

```
def getvarname (j) -> name
```

Obtains the name of a variable.

Parameters `j` (`int`) – Index of a variable. (input)

Return `name` (`str`) – Returns the required name.

Groups *Names*, *Problem data - linear part*, *Problem data - variables*, *Inspecting the task*

`Task.getvarnameindex`

```
def getvarnameindex (somename) -> asgn, index
```

Checks whether the name `somename` has been assigned to any variable. If so, the index of the variable is reported.

Parameters `somename` (`str`) – The name which should be checked. (input)

Return

- `asgn` (`int`) – Is non-zero if the name `somename` is assigned to a variable.
- `index` (`int`) – If the name `somename` is assigned to a variable, then `index` is the index of the variable.

Groups *Names*, *Problem data - linear part*, *Problem data - variables*, *Inspecting the task*

Task.getvarnamelen

```
def getvarnamelen (i) -> len
```

Obtains the length of the name of a variable.

Parameters i (int) – Index of a variable. (input)

Return len (int) – Returns the length of the indicated name.

Groups *Names*, *Problem data - linear part*, *Problem data - variables*, *Inspecting the task*

Task.getvartype

```
def getvartype (j) -> vartype
```

Gets the variable type of one variable.

Parameters j (int) – Index of the variable. (input)

Return vartype (*mosek.variabletype*) – Variable type of the j-th variable.

Groups *Inspecting the task*, *Problem data - variables*

Task.getvartypelist

```
def getvartypelist (subj, vartype)
```

Obtains the variable type of one or more variables. Upon return vartype[k] is the variable type of variable subj[k].

Parameters

- subj (int[]) – A list of variable indexes. (input)
- vartype (*mosek.variabletype* []) – The variables types corresponding to the variables specified by subj. (output)

Groups *Inspecting the task*, *Problem data - variables*

Task.getxc

```
def getxc (whichsol, xc)
```

Obtains the x^c vector for a solution.

Parameters

- whichsol (*mosek.soltype*) – Selects a solution. (input)
- xc (float[]) – Primal constraint solution. (output)

Groups *Solution - primal*

Task.getxcslice

```
def getxcslice (whichsol, first, last, xc)
```

Obtains a slice of the x^c vector for a solution.

Parameters

- whichsol (*mosek.soltype*) – Selects a solution. (input)
- first (int) – First index in the sequence. (input)
- last (int) – Last index plus 1 in the sequence. (input)
- xc (float[]) – Primal constraint solution. (output)

Groups *Solution - primal*

`Task.getxx`

```
def getxx (whichsol, xx)
```

Obtains the x^x vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `xx` (`float[]`) – Primal variable solution. (output)

Groups *Solution - primal*

`Task.getxxslice`

```
def getxxslice (whichsol, first, last, xx)
```

Obtains a slice of the x^x vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (`int`) – First index in the sequence. (input)
- `last` (`int`) – Last index plus 1 in the sequence. (input)
- `xx` (`float[]`) – Primal variable solution. (output)

Groups *Solution - primal*

`Task.gety`

```
def gety (whichsol, y)
```

Obtains the y vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `y` (`float[]`) – Vector of dual variables corresponding to the constraints. (output)

Groups *Solution - dual*

`Task.getyslice`

```
def getyslice (whichsol, first, last, y)
```

Obtains a slice of the y vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (`int`) – First index in the sequence. (input)
- `last` (`int`) – Last index plus 1 in the sequence. (input)
- `y` (`float[]`) – Vector of dual variables corresponding to the constraints. (output)

Groups *Solution - dual*

`Task.initbasissolve`

```
def initbasissolve (basis)
```

Prepare a task for use with the *Task.solvewithbasis* function.

This function should be called

- immediately before the first call to *Task.solvewithbasis*, and
- immediately before any subsequent call to *Task.solvewithbasis* if the task has been modified.

If the basis is singular i.e. not invertible, then the error *rescode.err_basis_singular* is reported.

Parameters `basis` (`int[]`) – The array of basis indexes to use. The array is interpreted as follows: If $\texttt{basis}[i] \leq numcon - 1$, then $x^c_{\texttt{basis}[i]}$ is in the basis at position i, otherwise $x_{\texttt{basis}[i]-\texttt{numcon}}$ is in the basis at position i. (output)

Groups *Solving systems with basis matrix*

`Task.inputdata`

```
def inputdata (maxnumcon, maxnumvar, c, cfix, aptrb, aptre, asub, aval, bkc, blc, buc,␣
↪bkx, blx, bux)
```

Input the linear part of an optimization problem.

The non-zeros of A are inputted column-wise in the format described in Section *Column or Row Ordered Sparse Matrix*.

For an explained code example see Section *Linear Optimization* and Section *Matrix Formats*.

Parameters

- `maxnumcon` (`int`) – Number of preallocated constraints in the optimization task. (input)
- `maxnumvar` (`int`) – Number of preallocated variables in the optimization task. (input)
- `c` (`float[]`) – Linear terms of the objective as a dense vector. The length is the number of variables. (input)
- `cfix` (`float`) – Fixed term in the objective. (input)
- `aptrb` (`int[]`) – Row or column start pointers. (input)
- `aptre` (`int[]`) – Row or column end pointers. (input)
- `asub` (`int[]`) – Coefficient subscripts. (input)
- `aval` (`float[]`) – Coefficient values. (input)
- `bkc` (*mosek.boundkey* `[]`) – Bound keys for the constraints. (input)
- `blc` (`float[]`) – Lower bounds for the constraints. (input)
- `buc` (`float[]`) – Upper bounds for the constraints. (input)
- `bkx` (*mosek.boundkey* `[]`) – Bound keys for the variables. (input)
- `blx` (`float[]`) – Lower bounds for the variables. (input)
- `bux` (`float[]`) – Upper bounds for the variables. (input)

Groups *Problem data - linear part*, *Problem data - bounds*, *Problem data - constraints*

`Task.isdouparname`

```
def isdouparname (parname) -> param
```

Checks whether `parname` is a valid double parameter name.

Parameters `parname` (`str`) – Parameter name. (input)

Return `param` (*mosek.dparam*) – Returns the parameter corresponding to the name, if one exists.

Groups *Parameters*, *Names*

`Task.isintparname`

```
def isintparname (parname) -> param
```

Checks whether **parname** is a valid integer parameter name.

Parameters parname (str) – Parameter name. (input)

Return param (*mosek.iparam*) – Returns the parameter corresponding to the name, if one exists.

Groups *Parameters*, *Names*

Task.isstrparname

```
def isstrparname (parname) -> param
```

Checks whether **parname** is a valid string parameter name.

Parameters parname (str) – Parameter name. (input)

Return param (*mosek.sparam*) – Returns the parameter corresponding to the name, if one exists.

Groups *Parameters*, *Names*

Task.linkfiletostream

```
def linkfiletostream (whichstream, filename, append)
```

Directs all output from a task stream **whichstream** to a file **filename**.

Parameters

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

Groups *Logging*

Task.onesolutionsummary

```
def onesolutionsummary (whichstream, whichsol)
```

Prints a short summary of a specified solution.

Parameters

- whichstream (*mosek.streamtype*) – Index of the stream. (input)
- whichsol (*mosek.soltype*) – Selects a solution. (input)

Groups *Logging*, *Solution information*

Task.optimize

```
def optimize () -> trmcode
```

Calls the optimizer. Depending on the problem type and the selected optimizer this will call one of the optimizers in **MOSEK**. By default the interior point optimizer will be selected for continuous problems. The optimizer may be selected manually by setting the parameter *iparam.optimizer*.

Return trmcode (*mosek.rescode*) – Is either *rescode.ok* or a termination response code.

Groups *Optimization*

Task.optimizermt

```
def optimizermt (server, port) -> trmcode
```

Offload the optimization task to a solver server defined by `server:port`. The call will block until a result is available or the connection closes.

If the string parameter *sparam.remote_access_token* is not blank, it will be passed to the server as authentication.

Parameters
- `server` (`str`) – Name or IP address of the solver server. (input)
- `port` (`str`) – Network port of the solver server. (input)

Return `trmcode` (*mosek.rescode*) – Is either *rescode.ok* or a termination response code.

Groups *Remote optimization*

`Task.optimizersummary`

```
def optimizersummary (whichstream)
```

Prints a short summary with optimizer statistics from last optimization.

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

Groups *Logging*

`Task.primalrepair`

```
def primalrepair (wlc, wuc, wlx, wux)
```

The function repairs a primal infeasible optimization problem by adjusting the bounds on the constraints and variables where the adjustment is computed as the minimal weighted sum of relaxations to the bounds on the constraints and variables. Observe the function only repairs the problem but does not solve it. If an optimal solution is required the problem should be optimized after the repair.

The function is applicable to linear and conic problems possibly with integer variables.

Observe that when computing the minimal weighted relaxation the termination tolerance specified by the parameters of the task is employed. For instance the parameter *iparam.mio_mode* can be used to make **MOSEK** ignore the integer constraints during the repair which usually leads to a much faster repair. However, the drawback is of course that the repaired problem may not have an integer feasible solution.

Note the function modifies the task in place. If this is not desired, then apply the function to a cloned task.

Parameters
- `wlc` (`float[]`) – $(w_l^c)_i$ is the weight associated with relaxing the lower bound on constraint i. If the weight is negative, then the lower bound is not relaxed. Moreover, if the argument is NULL, then all the weights are assumed to be 1. (input)
- `wuc` (`float[]`) – $(w_u^c)_i$ is the weight associated with relaxing the upper bound on constraint i. If the weight is negative, then the upper bound is not relaxed. Moreover, if the argument is NULL, then all the weights are assumed to be 1. (input)
- `wlx` (`float[]`) – $(w_l^x)_j$ is the weight associated with relaxing the lower bound on variable j. If the weight is negative, then the lower bound is not relaxed. Moreover, if the argument is NULL, then all the weights are assumed to be 1. (input)

- wux (float[]) – $(w_l^x)_i$ is the weight associated with relaxing the upper bound on variable j. If the weight is negative, then the upper bound is not relaxed. Moreover, if the argument is NULL, then all the weights are assumed to be 1. (input)

Groups *Infeasibility diagnostic*

Task.primalsensitivity

```
def primalsensitivity (subi, marki, subj, markj, leftpricei, rightpricei, leftrangei,␣
↪rightrangei, leftpricej, rightpricej, leftrangej, rightrangej)
```

Calculates sensitivity information for bounds on variables and constraints. For details on sensitivity analysis, the definitions of *shadow price* and *linearity interval* and an example see Section *Sensitivity Analysis*.

The type of sensitivity analysis to be performed (basis or optimal partition) is controlled by the parameter *iparam.sensitivity_type*.

Parameters

- subi (int[]) – Indexes of constraints to analyze. (input)
- marki (*mosek.mark* []) – The value of marki[i] indicates for which bound of constraint subi[i] sensitivity analysis is performed. If marki[i] = *mark.up* the upper bound of constraint subi[i] is analyzed, and if marki[i] = *mark.lo* the lower bound is analyzed. If subi[i] is an equality constraint, either *mark.lo* or *mark.up* can be used to select the constraint for sensitivity analysis. (input)
- subj (int[]) – Indexes of variables to analyze. (input)
- markj (*mosek.mark* []) – The value of markj[j] indicates for which bound of variable subj[j] sensitivity analysis is performed. If markj[j] = *mark.up* the upper bound of variable subj[j] is analyzed, and if markj[j] = *mark.lo* the lower bound is analyzed. If subj[j] is a fixed variable, either *mark.lo* or *mark.up* can be used to select the bound for sensitivity analysis. (input)
- leftpricei (float[]) – leftpricei[i] is the left shadow price for the bound marki[i] of constraint subi[i]. (output)
- rightpricei (float[]) – rightpricei[i] is the right shadow price for the bound marki[i] of constraint subi[i]. (output)
- leftrangei (float[]) – leftrangei[i] is the left range β_1 for the bound marki[i] of constraint subi[i]. (output)
- rightrangei (float[]) – rightrangei[i] is the right range β_2 for the bound marki[i] of constraint subi[i]. (output)
- leftpricej (float[]) – leftpricej[j] is the left shadow price for the bound markj[j] of variable subj[j]. (output)
- rightpricej (float[]) – rightpricej[j] is the right shadow price for the bound markj[j] of variable subj[j]. (output)
- leftrangej (float[]) – leftrangej[j] is the left range β_1 for the bound markj[j] of variable subj[j]. (output)
- rightrangej (float[]) – rightrangej[j] is the right range β_2 for the bound markj[j] of variable subj[j]. (output)

Groups *Sensitivity analysis*

Task.putacol

```
def putacol (j, subj, valj)
```

Change one column of the linear constraint matrix A. Resets all the elements in column j to zero and then sets

$$a_{\mathtt{subj}[k],\mathtt{j}} = \mathtt{valj}[k], \quad k = 0, \ldots, \mathtt{nzj} - 1.$$

Parameters

- j (int) – Index of a column in A. (input)
- subj (int[]) – Row indexes of non-zero values in column j of A. (input)
- valj (float[]) – New non-zero values of column j in A. (input)

Groups *Problem data - linear part*

Task.putacollist

```
def putacollist (sub, ptrb, ptre, asub, aval)
```

Change a set of columns in the linear constraint matrix A with data in sparse triplet format. The requested columns are set to zero and then updated with:

$$\begin{array}{ll} \texttt{for} & i = 0, \ldots, num - 1 \\ & a_{\texttt{asub}[k], \texttt{sub}[i]} = \texttt{aval}[k], \quad k = \texttt{ptrb}[i], \ldots, \texttt{ptre}[i] - 1. \end{array}$$

Parameters

- sub (int[]) – Indexes of columns that should be replaced, no duplicates. (input)
- ptrb (int[]) – Array of pointers to the first element in each column. (input)
- ptre (int[]) – Array of pointers to the last element plus one in each column. (input)
- asub (int[]) – Row indexes of new elements. (input)
- aval (float[]) – Coefficient values. (input)

Groups *Problem data - linear part*

Task.putacolslice

```
def putacolslice (first, last, ptrb, ptre, asub, aval)
```

Change a slice of columns in the linear constraint matrix A with data in sparse triplet format. The requested columns are set to zero and then updated with:

$$\begin{array}{ll} \texttt{for} & i = \texttt{first}, \ldots, \texttt{last} - 1 \\ & a_{\texttt{asub}[k], i} = \texttt{aval}[k], \quad k = \texttt{ptrb}[i], \ldots, \texttt{ptre}[i] - 1. \end{array}$$

Parameters

- first (int) – First column in the slice. (input)
- last (int) – Last column plus one in the slice. (input)
- ptrb (int[]) – Array of pointers to the first element in each column. (input)
- ptre (int[]) – Array of pointers to the last element plus one in each column. (input)
- asub (int[]) – Row indexes of new elements. (input)
- aval (float[]) – Coefficient values. (input)

Groups *Problem data - linear part*

Task.putaij

```
def putaij (i, j, aij)
```

Changes a coefficient in the linear coefficient matrix A using the method

$$a_{i,j} = \texttt{aij}.$$

Parameters

- i (int) – Constraint (row) index. (input)
- j (int) – Variable (column) index. (input)

- aij (float) – New coefficient for $a_{i,j}$. (input)

Groups *Problem data - linear part*

`Task.putaijlist`

```
def putaijlist (subi, subj, valij)
```

Changes one or more coefficients in A using the method

$$a_{\mathtt{subi[k]},\mathtt{subj[k]}} = \mathtt{valij[k]}, \quad k = 0, \ldots, num - 1.$$

Duplicates are not allowed.

Parameters

- subi (int[]) – Constraint (row) indices. (input)
- subj (int[]) – Variable (column) indices. (input)
- valij (float[]) – New coefficient values for $a_{i,j}$. (input)

Groups *Problem data - linear part*

`Task.putarow`

```
def putarow (i, subi, vali)
```

Change one row of the linear constraint matrix A. Resets all the elements in row i to zero and then sets

$$a_{\mathtt{i},\mathtt{subi}[k]} = \mathtt{vali}[k], \quad k = 0, \ldots, \mathtt{nzi} - 1.$$

Parameters

- i (int) – Index of a row in A. (input)
- subi (int[]) – Column indexes of non-zero values in row i of A. (input)
- vali (float[]) – New non-zero values of row i in A. (input)

Groups *Problem data - linear part*

`Task.putarowlist`

```
def putarowlist (sub, ptrb, ptre, asub, aval)
```

Change a set of rows in the linear constraint matrix A with data in sparse triplet format. The requested rows are set to zero and then updated with:

$$\begin{array}{ll} \mathtt{for} & i = 0, \ldots, num - 1 \\ & a_{\mathtt{sub}[i],\mathtt{asub}[k]} = \mathtt{aval}[k], \quad k = \mathtt{ptrb}[i], \ldots, \mathtt{ptre}[i] - 1. \end{array}$$

Parameters

- sub (int[]) – Indexes of rows that should be replaced, no duplicates. (input)
- ptrb (int[]) – Array of pointers to the first element in each row. (input)
- ptre (int[]) – Array of pointers to the last element plus one in each row. (input)
- asub (int[]) – Column indexes of new elements. (input)
- aval (float[]) – Coefficient values. (input)

Groups *Problem data - linear part*

`Task.putarowslice`

```
def putarowslice (first, last, ptrb, ptre, asub, aval)
```

Change a slice of rows in the linear constraint matrix A with data in sparse triplet format. The requested columns are set to zero and then updated with:

$$\begin{array}{rl} \text{for} & i = \texttt{first}, \ldots, \texttt{last} - 1 \\ & a_{\texttt{sub}[i], \texttt{asub}[k]} = \texttt{aval}[k], \quad k = \texttt{ptrb}[i], \ldots, \texttt{ptre}[i] - 1. \end{array}$$

Parameters

- `first` (`int`) – First row in the slice. (input)
- `last` (`int`) – Last row plus one in the slice. (input)
- `ptrb` (`int[]`) – Array of pointers to the first element in each row. (input)
- `ptre` (`int[]`) – Array of pointers to the last element plus one in each row. (input)
- `asub` (`int[]`) – Column indexes of new elements. (input)
- `aval` (`float[]`) – Coefficient values. (input)

Groups *Problem data - linear part*

`Task.putatruncatetol`

```
def putatruncatetol (tolzero)
```

Truncates (sets to zero) all elements in A that satisfy

$$|a_{i,j}| \leq \texttt{tolzero}.$$

Parameters `tolzero` (`float`) – Truncation tolerance. (input)

Groups *Problem data - linear part*

`Task.putbarablocktriplet`

```
def putbarablocktriplet (num, subi, subj, subk, subl, valijkl)
```

Inputs the $\overline{A}$ matrix in block triplet form.

Parameters

- `num` (`int`) – Number of elements in the block triplet form. (input)
- `subi` (`int[]`) – Constraint index. (input)
- `subj` (`int[]`) – Symmetric matrix variable index. (input)
- `subk` (`int[]`) – Block row index. (input)
- `subl` (`int[]`) – Block column index. (input)
- `valijkl` (`float[]`) – The numerical value associated with each block triplet. (input)

Groups *Problem data - semidefinite*

`Task.putbaraij`

```
def putbaraij (i, j, sub, weights)
```

This function sets one element in the $\overline{A}$ matrix.

Each element in the $\overline{A}$ matrix is a weighted sum of symmetric matrices from the symmetric matrix storage E, so $\overline{A}_{ij}$ is a symmetric matrix. By default all elements in $\overline{A}$ are 0, so only non-zero elements need be added. Setting the same element again will overwrite the earlier entry.

The symmetric matrices from E are defined separately using the function *`Task.appendsparsesymmat`*.

Parameters

- i (int) – Row index of $\overline{A}$. (input)
- j (int) – Column index of $\overline{A}$. (input)
- sub (int[]) – Indices in E of the matrices appearing in the weighted sum for $\overline{A}_{ij}$. (input)
- weights (float[]) – weights[k] is the coefficient of the sub[k]-th element of E in the weighted sum forming $\overline{A}_{ij}$. (input)

Groups *Problem data - semidefinite*

Task.putbaraijlist

```
def putbaraijlist (subi, subj, alphaptrb, alphaptre, matidx, weights)
```

This function sets a list of elements in the $\overline{A}$ matrix.

Each element in the $\overline{A}$ matrix is a weighted sum of symmetric matrices from the symmetric matrix storage E, so $\overline{A}_{ij}$ is a symmetric matrix. By default all elements in $\overline{A}$ are 0, so only non-zero elements need be added. Setting the same element again will overwrite the earlier entry.

The symmetric matrices from E are defined separately using the function *Task.appendsparsesymmat*.

Parameters

- subi (int[]) – Row index of $\overline{A}$. (input)
- subj (int[]) – Column index of $\overline{A}$. (input)
- alphaptrb (int[]) – Start entries for terms in the weighted sum that forms $\overline{A}_{ij}$. (input)
- alphaptre (int[]) – End entries for terms in the weighted sum that forms $\overline{A}_{ij}$. (input)
- matidx (int[]) – Indices in E of the matrices appearing in the weighted sum for $\overline{A}_{ij}$. (input)
- weights (float[]) – weights[k] is the coefficient of the sub[k]-th element of E in the weighted sum forming $\overline{A}_{ij}$. (input)

Groups *Problem data - semidefinite*

Task.putbararowlist

```
def putbararowlist (subi, ptrb, ptre, subj, nummat, matidx, weights)
```

This function replaces a list of rows in the $\overline{A}$ matrix.

Parameters

- subi (int[]) – Row indexes of $\overline{A}$. (input)
- ptrb (int[]) – Start of rows in $\overline{A}$. (input)
- ptre (int[]) – End of rows in $\overline{A}$. (input)
- subj (int[]) – Column index of $\overline{A}$. (input)
- nummat (int[]) – Number of entries in weighted sum of matrixes. (input)
- matidx (int[]) – Matrix indexes for weighted sum of matrixes. (input)
- weights (float[]) – Weights for weighted sum of matrixes. (input)

Groups *Problem data - semidefinite*

Task.putbarcblocktriplet

```
def putbarcblocktriplet (num, subj, subk, subl, valjkl)
```

Inputs the $\overline{C}$ matrix in block triplet form.

Parameters

- `num` (`int`) – Number of elements in the block triplet form. (input)
- `subj` (`int[]`) – Symmetric matrix variable index. (input)
- `subk` (`int[]`) – Block row index. (input)
- `subl` (`int[]`) – Block column index. (input)
- `valjkl` (`float[]`) – The numerical value associated with each block triplet. (input)

Groups *Problem data - semidefinite*

`Task.putbarcj`

```
def putbarcj (j, sub, weights)
```

This function sets one entry in the $\overline{C}$ vector.

Each element in the $\overline{C}$ vector is a weighted sum of symmetric matrices from the symmetric matrix storage E, so $\overline{C}_j$ is a symmetric matrix. By default all elements in $\overline{C}$ are 0, so only non-zero elements need be added. Setting the same element again will overwrite the earlier entry.

The symmetric matrices from E are defined separately using the function *`Task.appendsparsesymmat`*.

Parameters

- `j` (`int`) – Index of the element in $\overline{C}$ that should be changed. (input)
- `sub` (`int[]`) – Indices in E of matrices appearing in the weighted sum for $\overline{C}_j$ (input)
- `weights` (`float[]`) – `weights[k]` is the coefficient of the `sub[k]`-th element of E in the weighted sum forming $\overline{C}_j$. (input)

Groups *Problem data - semidefinite*, *Problem data - objective*

`Task.putbarsj`

```
def putbarsj (whichsol, j, barsj)
```

Sets the dual solution for a semidefinite variable.

Parameters

- `whichsol` (*`mosek.soltype`*) – Selects a solution. (input)
- `j` (`int`) – Index of the semidefinite variable. (input)
- `barsj` (`float[]`) – Value of $\overline{S}_j$. Format as in *`Task.getbarsj`*. (input)

Groups *Solution - semidefinite*

`Task.putbarvarname`

```
def putbarvarname (j, name)
```

Sets the name of a semidefinite variable.

Parameters

- `j` (`int`) – Index of the variable. (input)
- `name` (`str`) – The variable name. (input)

Groups *Names*, *Problem data - semidefinite*

`Task.putbarxj`

```
def putbarxj (whichsol, j, barxj)
```

Sets the primal solution for a semidefinite variable.

Parameters

- whichsol (*mosek.soltype*) – Selects a solution. (input)
- j (int) – Index of the semidefinite variable. (input)
- barxj (float[]) – Value of $\overline{X}_j$. Format as in *Task.getbarxj*. (input)

Groups *Solution - semidefinite*

Task.putcfix

```
def putcfix (cfix)
```

Replaces the fixed term in the objective by a new one.

Parameters cfix (float) – Fixed term in the objective. (input)

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

Task.putcj

```
def putcj (j, cj)
```

Modifies one coefficient in the linear objective vector c, i.e.

$$c_{\mathtt{j}} = \mathtt{cj}.$$

If the absolute value exceeds *dparam.data_tol_c_huge* an error is generated. If the absolute value exceeds *dparam.data_tol_cj_large*, a warning is generated, but the coefficient is inputted as specified.

Parameters

- j (int) – Index of the variable for which c should be changed. (input)
- cj (float) – New value of c_j. (input)

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

Task.putclist

```
def putclist (subj, val)
```

Modifies the coefficients in the linear term c in the objective using the principle

$$c_{\mathtt{subj[t]}} = \mathtt{val}[\mathtt{t}], \quad t = 0, \ldots, num - 1.$$

If a variable index is specified multiple times in subj only the last entry is used. Data checks are performed as in *Task.putcj*.

Parameters

- subj (int[]) – Indices of variables for which the coefficient in c should be changed. (input)
- val (float[]) – New numerical values for coefficients in c that should be modified. (input)

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

Task.putconbound

```
def putconbound (i, bkc, blc, buc)
```

Changes the bounds for one constraint.

If the bound value specified is numerically larger than *dparam.data_tol_bound_inf* it is considered infinite and the bound key is changed accordingly. If a bound value is numerically larger than *dparam.data_tol_bound_wrn*, a warning will be displayed, but the bound is inputted as specified.

Parameters

- i (int) – Index of the constraint. (input)
- bkc (*mosek.boundkey*) – New bound key. (input)
- blc (float) – New lower bound. (input)
- buc (float) – New upper bound. (input)

Groups *Problem data - linear part*, *Problem data - constraints*, *Problem data - bounds*

Task.putconboundlist

```
def putconboundlist (sub, bkc, blc, buc)
```

Changes the bounds for a list of constraints. If multiple bound changes are specified for a constraint, then only the last change takes effect. Data checks are performed as in *Task.putconbound*.

Parameters

- sub (int[]) – List of constraint indexes. (input)
- bkc (*mosek.boundkey* []) – Bound keys for the constraints. (input)
- blc (float[]) – Lower bounds for the constraints. (input)
- buc (float[]) – Upper bounds for the constraints. (input)

Groups *Problem data - linear part*, *Problem data - constraints*, *Problem data - bounds*

Task.putconboundlistconst

```
def putconboundlistconst (sub, bkc, blc, buc)
```

Changes the bounds for one or more constraints. Data checks are performed as in *Task.putconbound*.

Parameters

- sub (int[]) – List of constraint indexes. (input)
- bkc (*mosek.boundkey*) – New bound key for all constraints in the list. (input)
- blc (float) – New lower bound for all constraints in the list. (input)
- buc (float) – New upper bound for all constraints in the list. (input)

Groups *Problem data - linear part*, *Problem data - constraints*, *Problem data - bounds*

Task.putconboundslice

```
def putconboundslice (first, last, bkc, blc, buc)
```

Changes the bounds for a slice of the constraints. Data checks are performed as in *Task.putconbound*.

Parameters

- first (int) – First index in the sequence. (input)
- last (int) – Last index plus 1 in the sequence. (input)
- bkc (*mosek.boundkey* []) – Bound keys for the constraints. (input)
- blc (float[]) – Lower bounds for the constraints. (input)
- buc (float[]) – Upper bounds for the constraints. (input)

Groups *Problem data - linear part*, *Problem data - constraints*, *Problem data - bounds*

Task.putconboundsliceconst

```
def putconboundsliceconst (first, last, bkc, blc, buc)
```

Changes the bounds for a slice of the constraints. Data checks are performed as in *Task.putconbound*.

Parameters

- first (int) – First index in the sequence. (input)
- last (int) – Last index plus 1 in the sequence. (input)
- bkc (*mosek.boundkey*) – New bound key for all constraints in the slice. (input)
- blc (float) – New lower bound for all constraints in the slice. (input)
- buc (float) – New upper bound for all constraints in the slice. (input)

Groups *Problem data - linear part*, *Problem data - constraints*, *Problem data - bounds*

Task.putcone

```
def putcone (k, ct, conepar, submem)
```

Replaces a conic constraint.

Parameters

- k (int) – Index of the cone. (input)
- ct (*mosek.conetype*) – Specifies the type of the cone. (input)
- conepar (float) – For the power cone it denotes the exponent alpha. For other cone types it is unused and can be set to 0. (input)
- submem (int[]) – Variable subscripts of the members in the cone. (input)

Groups *Problem data - cones*

Task.putconename

```
def putconename (j, name)
```

Sets the name of a cone.

Parameters

- j (int) – Index of the cone. (input)
- name (str) – The name of the cone. (input)

Groups *Names*, *Problem data - cones*

Task.putconname

```
def putconname (i, name)
```

Sets the name of a constraint.

Parameters

- i (int) – Index of the constraint. (input)
- name (str) – The name of the constraint. (input)

Groups *Names*, *Problem data - constraints*, *Problem data - linear part*

Task.putconsolutioni

```
def putconsolutioni (i, whichsol, sk, x, sl, su)
```

Sets the primal and dual solution information for a single constraint.

Parameters

- i (int) – Index of the constraint. (input)
- whichsol (*mosek.soltype*) – Selects a solution. (input)
- sk (*mosek.stakey*) – Status key of the constraint. (input)
- x (float) – Primal solution value of the constraint. (input)

- sl (float) – Solution value of the dual variable associated with the lower bound. (input)
- su (float) – Solution value of the dual variable associated with the upper bound. (input)

Groups *Solution information*, *Solution - primal*, *Solution - dual*

`Task.putcslice`

```
def putcslice (first, last, slice)
```

Modifies a slice in the linear term c in the objective using the principle

$$c_{\mathtt{j}} = \mathtt{slice}[\mathtt{j} - \mathtt{first}], \quad j = first, .., last - 1$$

Data checks are performed as in *Task.putcj*.

Parameters

- first (int) – First element in the slice of c. (input)
- last (int) – Last element plus 1 of the slice in c to be changed. (input)
- slice (float[]) – New numerical values for coefficients in c that should be modified. (input)

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

`Task.putdouparam`

```
def putdouparam (param, parvalue)
```

Sets the value of a double parameter.

Parameters

- param (*mosek.dparam*) – Which parameter. (input)
- parvalue (float) – Parameter value. (input)

Groups *Parameters*

`Task.putintparam`

```
def putintparam (param, parvalue)
```

Sets the value of an integer parameter.

Parameters

- param (*mosek.iparam*) – Which parameter. (input)
- parvalue (int) – Parameter value. (input)

Groups *Parameters*

`Task.putmaxnumanz`

```
def putmaxnumanz (maxnumanz)
```

Sets the number of preallocated non-zero entries in A.

MOSEK stores only the non-zero elements in the linear coefficient matrix A and it cannot predict how much storage is required to store A. Using this function it is possible to specify the number of non-zeros to preallocate for storing A.

If the number of non-zeros in the problem is known, it is a good idea to set `maxnumanz` slightly larger than this number, otherwise a rough estimate can be used. In general, if A is inputted in many small chunks, setting this value may speed up the data input phase.

It is not mandatory to call this function, since **MOSEK** will reallocate internal structures whenever it is necessary.

The function call has no effect if both `maxnumcon` and `maxnumvar` are zero.

Parameters `maxnumanz` (`int`) – Number of preallocated non-zeros in A. (input)

Groups *Environment and task management*, *Problem data - semidefinite*

`Task.putmaxnumbarvar`

```
def putmaxnumbarvar (maxnumbarvar)
```

Sets the number of preallocated symmetric matrix variables in the optimization task. When this number of variables is reached **MOSEK** will automatically allocate more space for variables.

It is not mandatory to call this function. It only gives a hint about the amount of data to preallocate for efficiency reasons.

Please note that `maxnumbarvar` must be larger than the current number of symmetric matrix variables in the task.

Parameters `maxnumbarvar` (`int`) – Number of preallocated symmetric matrix variables. (input)

Groups *Environment and task management*, *Problem data - semidefinite*

`Task.putmaxnumcon`

```
def putmaxnumcon (maxnumcon)
```

Sets the number of preallocated constraints in the optimization task. When this number of constraints is reached **MOSEK** will automatically allocate more space for constraints.

It is never mandatory to call this function, since **MOSEK** will reallocate any internal structures whenever it is required.

Please note that `maxnumcon` must be larger than the current number of constraints in the task.

Parameters `maxnumcon` (`int`) – Number of preallocated constraints in the optimization task. (input)

Groups *Environment and task management*, *Problem data - constraints*

`Task.putmaxnumcone`

```
def putmaxnumcone (maxnumcone)
```

Sets the number of preallocated conic constraints in the optimization task. When this number of conic constraints is reached **MOSEK** will automatically allocate more space for conic constraints.

It is not mandatory to call this function, since **MOSEK** will reallocate any internal structures whenever it is required.

Please note that `maxnumcon` must be larger than the current number of conic constraints in the task.

Parameters `maxnumcone` (`int`) – Number of preallocated conic constraints in the optimization task. (input)

Groups *Environment and task management*, *Problem data - cones*

`Task.putmaxnumqnz`

```
def putmaxnumqnz (maxnumqnz)
```

Sets the number of preallocated non-zero entries in quadratic terms.

MOSEK stores only the non-zero elements in Q. Therefore, **MOSEK** cannot predict how much storage is required to store Q. Using this function it is possible to specify the number non-zeros to preallocate for storing Q (both objective and constraints).

It may be advantageous to reserve more non-zeros for Q than actually needed since it may improve the internal efficiency of **MOSEK**, however, it is never worthwhile to specify more than the double of the anticipated number of non-zeros in Q.

It is not mandatory to call this function, since **MOSEK** will reallocate internal structures whenever it is necessary.

Parameters `maxnumqnz` (`int`) – Number of non-zero elements preallocated in quadratic coefficient matrices. (input)

Groups *Environment and task management*, *Problem data - quadratic part*

`Task.putmaxnumvar`

```
def putmaxnumvar (maxnumvar)
```

Sets the number of preallocated variables in the optimization task. When this number of variables is reached **MOSEK** will automatically allocate more space for variables.

It is not mandatory to call this function. It only gives a hint about the amount of data to preallocate for efficiency reasons.

Please note that `maxnumvar` must be larger than the current number of variables in the task.

Parameters `maxnumvar` (`int`) – Number of preallocated variables in the optimization task. (input)

Groups *Environment and task management*, *Problem data - variables*

`Task.putnadouparam`

```
def putnadouparam (paramname, parvalue)
```

Sets the value of a named double parameter.

Parameters

- `paramname` (`str`) – Name of a parameter. (input)
- `parvalue` (`float`) – Parameter value. (input)

Groups *Parameters*

`Task.putnaintparam`

```
def putnaintparam (paramname, parvalue)
```

Sets the value of a named integer parameter.

Parameters

- `paramname` (`str`) – Name of a parameter. (input)
- `parvalue` (`int`) – Parameter value. (input)

Groups *Parameters*

`Task.putnastrparam`

```
def putnastrparam (paramname, parvalue)
```

Sets the value of a named string parameter.

Parameters

- `paramname` (`str`) – Name of a parameter. (input)
- `parvalue` (`str`) – Parameter value. (input)

Groups *Parameters*

`Task.putobjname`

```
def putobjname (objname)
```

Assigns a new name to the objective.

Parameters `objname` (`str`) – Name of the objective. (input)

Groups *Problem data - linear part*, *Names*, *Problem data - objective*

`Task.putobjsense`

```
def putobjsense (sense)
```

Sets the objective sense of the task.

Parameters `sense` (*mosek.objsense*) – The objective sense of the task. The values *objsense.maximize* and *objsense.minimize* mean that the problem is maximized or minimized respectively. (input)

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

`Task.putparam`

```
def putparam (parname, parvalue)
```

Checks if `parname` is valid parameter name. If it is, the parameter is assigned the value specified by `parvalue`.

Parameters

- `parname` (`str`) – Parameter name. (input)
- `parvalue` (`str`) – Parameter value. (input)

Groups *Parameters*

`Task.putqcon`

```
def putqcon (qcsubk, qcsubi, qcsubj, qcval)
```

Replace all quadratic entries in the constraints. The list of constraints has the form

$$l_k^c \leq \frac{1}{2} \sum_{i=0}^{numvar-1} \sum_{j=0}^{numvar-1} q_{ij}^k x_i x_j + \sum_{j=0}^{numvar-1} a_{kj} x_j \leq u_k^c, \quad k = 0, \ldots, m-1.$$

This function sets all the quadratic terms to zero and then performs the update:

$$q_{\mathtt{qcsubi[t]},\mathtt{qcsubj[t]}}^{\mathtt{qcsubk[t]}} = q_{\mathtt{qcsubj[t]},\mathtt{qcsubi[t]}}^{\mathtt{qcsubk[t]}} = q_{\mathtt{qcsubj[t]},\mathtt{qcsubi[t]}}^{\mathtt{qcsubk[t]}} + \mathtt{qcval[t]},$$

for $t = 0, \ldots, numqcnz - 1$.

Please note that:

- For large problems it is essential for the efficiency that the function *Task.putmaxnumqnz* is employed to pre-allocate space.
- Only the lower triangular parts should be specified because the Q matrices are symmetric. Specifying entries where $i < j$ will result in an error.
- Only non-zero elements should be specified.

- The order in which the non-zero elements are specified is insignificant.
- Duplicate elements are added together as shown above. Hence, it is usually not recommended to specify the same entry multiple times.

For a code example see Section *Quadratic Optimization*

Parameters

- qcsubk (int[]) – Constraint subscripts for quadratic coefficients. (input)
- qcsubi (int[]) – Row subscripts for quadratic constraint matrix. (input)
- qcsubj (int[]) – Column subscripts for quadratic constraint matrix. (input)
- qcval (float[]) – Quadratic constraint coefficient values. (input)

Groups *Problem data - quadratic part*

Task.putqconk

```
def putqconk (k, qcsubi, qcsubj, qcval)
```

Replaces all the quadratic entries in one constraint. This function performs the same operations as *Task.putqcon* but only with respect to constraint number k and it does not modify the other constraints. See the description of *Task.putqcon* for definitions and important remarks.

Parameters

- k (int) – The constraint in which the new Q elements are inserted. (input)
- qcsubi (int[]) – Row subscripts for quadratic constraint matrix. (input)
- qcsubj (int[]) – Column subscripts for quadratic constraint matrix. (input)
- qcval (float[]) – Quadratic constraint coefficient values. (input)

Groups *Problem data - quadratic part*

Task.putqobj

```
def putqobj (qosubi, qosubj, qoval)
```

Replace all quadratic terms in the objective. If the objective has the form

$$\frac{1}{2}\sum_{i=0}^{numvar-1}\sum_{j=0}^{numvar-1} q_{ij}^{o} x_i x_j + \sum_{j=0}^{numvar-1} c_j x_j + c^f$$

then this function sets all the quadratic terms to zero and then performs the update:

$$q^{o}_{\mathtt{qosubi[t]},\mathtt{qosubj[t]}} = q^{o}_{\mathtt{qosubj[t]},\mathtt{qosubi[t]}} = q^{o}_{\mathtt{qosubj[t]},\mathtt{qosubi[t]}} + \mathtt{qoval[t]},$$

for $t = 0, \ldots, numqonz - 1$.

See the description of *Task.putqcon* for important remarks and example.

Parameters

- qosubi (int[]) – Row subscripts for quadratic objective coefficients. (input)
- qosubj (int[]) – Column subscripts for quadratic objective coefficients. (input)
- qoval (float[]) – Quadratic objective coefficient values. (input)

Groups *Problem data - quadratic part*, *Problem data - objective*

Task.putqobjij

```
def putqobjij (i, j, qoij)
```

Replaces one coefficient in the quadratic term in the objective. The function performs the assignment

$$q^o_{ij} = q^o_{ji} = \mathtt{qoij}.$$

Only the elements in the lower triangular part are accepted. Setting q_{ij} with $j > i$ will cause an error.

Please note that replacing all quadratic elements one by one is more computationally expensive than replacing them all at once. Use *Task.putqobj* instead whenever possible.

Parameters

- `i` (`int`) – Row index for the coefficient to be replaced. (input)
- `j` (`int`) – Column index for the coefficient to be replaced. (input)
- `qoij` (`float`) – The new value for q^o_{ij}. (input)

Groups *Problem data - quadratic part*, *Problem data - objective*

`Task.putskc`

```
def putskc (whichsol, skc)
```

Sets the status keys for the constraints.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `skc` (*mosek.stakey* `[]`) – Status keys for the constraints. (input)

Groups *Solution information*

`Task.putskcslice`

```
def putskcslice (whichsol, first, last, skc)
```

Sets the status keys for a slice of the constraints.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (`int`) – First index in the sequence. (input)
- `last` (`int`) – Last index plus 1 in the sequence. (input)
- `skc` (*mosek.stakey* `[]`) – Status keys for the constraints. (input)

Groups *Solution information*

`Task.putskx`

```
def putskx (whichsol, skx)
```

Sets the status keys for the scalar variables.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `skx` (*mosek.stakey* `[]`) – Status keys for the variables. (input)

Groups *Solution information*

`Task.putskxslice`

```
def putskxslice (whichsol, first, last, skx)
```

Sets the status keys for a slice of the variables.

Parameters

- whichsol (*mosek.soltype*) – Selects a solution. (input)
- first (int) – First index in the sequence. (input)
- last (int) – Last index plus 1 in the sequence. (input)
- skx (*mosek.stakey* []) – Status keys for the variables. (input)

Groups *Solution information*

Task.putslc

```
def putslc (whichsol, slc)
```

Sets the s_l^c vector for a solution.

Parameters

- whichsol (*mosek.soltype*) – Selects a solution. (input)
- slc (float[]) – Dual variables corresponding to the lower bounds on the constraints. (input)

Groups *Solution - dual*

Task.putslcslice

```
def putslcslice (whichsol, first, last, slc)
```

Sets a slice of the s_l^c vector for a solution.

Parameters

- whichsol (*mosek.soltype*) – Selects a solution. (input)
- first (int) – First index in the sequence. (input)
- last (int) – Last index plus 1 in the sequence. (input)
- slc (float[]) – Dual variables corresponding to the lower bounds on the constraints. (input)

Groups *Solution - dual*

Task.putslx

```
def putslx (whichsol, slx)
```

Sets the s_l^x vector for a solution.

Parameters

- whichsol (*mosek.soltype*) – Selects a solution. (input)
- slx (float[]) – Dual variables corresponding to the lower bounds on the variables. (input)

Groups *Solution - dual*

Task.putslxslice

```
def putslxslice (whichsol, first, last, slx)
```

Sets a slice of the s_l^x vector for a solution.

Parameters

- whichsol (*mosek.soltype*) – Selects a solution. (input)
- first (int) – First index in the sequence. (input)
- last (int) – Last index plus 1 in the sequence. (input)

- slx (float[]) – Dual variables corresponding to the lower bounds on the variables. (input)

Groups *Solution - dual*

Task.putsnx

```
def putsnx (whichsol, sux)
```

Sets the s_n^x vector for a solution.

Parameters

- whichsol (*mosek.soltype*) – Selects a solution. (input)
- sux (float[]) – Dual variables corresponding to the upper bounds on the variables. (input)

Groups *Solution - dual*

Task.putsnxslice

```
def putsnxslice (whichsol, first, last, snx)
```

Sets a slice of the s_n^x vector for a solution.

Parameters

- whichsol (*mosek.soltype*) – Selects a solution. (input)
- first (int) – First index in the sequence. (input)
- last (int) – Last index plus 1 in the sequence. (input)
- snx (float[]) – Dual variables corresponding to the conic constraints on the variables. (input)

Groups *Solution - dual*

Task.putsolution

```
def putsolution (whichsol, skc, skx, skn, xc, xx, y, slc, suc, slx, sux, snx)
```

Inserts a solution into the task.

Parameters

- whichsol (*mosek.soltype*) – Selects a solution. (input)
- skc (*mosek.stakey* []) – Status keys for the constraints. (input)
- skx (*mosek.stakey* []) – Status keys for the variables. (input)
- skn (*mosek.stakey* []) – Status keys for the conic constraints. (input)
- xc (float[]) – Primal constraint solution. (input)
- xx (float[]) – Primal variable solution. (input)
- y (float[]) – Vector of dual variables corresponding to the constraints. (input)
- slc (float[]) – Dual variables corresponding to the lower bounds on the constraints. (input)
- suc (float[]) – Dual variables corresponding to the upper bounds on the constraints. (input)
- slx (float[]) – Dual variables corresponding to the lower bounds on the variables. (input)
- sux (float[]) – Dual variables corresponding to the upper bounds on the variables. (input)
- snx (float[]) – Dual variables corresponding to the conic constraints on the variables. (input)

Groups *Solution information*, *Solution - primal*, *Solution - dual*

Task.putsolutionyi

```
def putsolutionyi (i, whichsol, y)
```

Inputs the dual variable of a solution.

Parameters
- `i` (`int`) – Index of the dual variable. (input)
- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `y` (`float`) – Solution value of the dual variable. (input)

Groups *Solution information*, *Solution - dual*

Task.putstrparam

```
def putstrparam (param, parvalue)
```

Sets the value of a string parameter.

Parameters
- `param` (*mosek.sparam*) – Which parameter. (input)
- `parvalue` (`str`) – Parameter value. (input)

Groups *Parameters*

Task.putsuc

```
def putsuc (whichsol, suc)
```

Sets the s_u^c vector for a solution.

Parameters
- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `suc` (`float[]`) – Dual variables corresponding to the upper bounds on the constraints. (input)

Groups *Solution - dual*

Task.putsucslice

```
def putsucslice (whichsol, first, last, suc)
```

Sets a slice of the s_u^c vector for a solution.

Parameters
- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (`int`) – First index in the sequence. (input)
- `last` (`int`) – Last index plus 1 in the sequence. (input)
- `suc` (`float[]`) – Dual variables corresponding to the upper bounds on the constraints. (input)

Groups *Solution - dual*

Task.putsux

```
def putsux (whichsol, sux)
```

Sets the s_u^x vector for a solution.

Parameters

- whichsol (*mosek.soltype*) – Selects a solution. (input)
- sux (float[]) – Dual variables corresponding to the upper bounds on the variables. (input)

Groups *Solution - dual*

Task.putsuxslice

```
def putsuxslice (whichsol, first, last, sux)
```

Sets a slice of the s_u^x vector for a solution.

Parameters

- whichsol (*mosek.soltype*) – Selects a solution. (input)
- first (int) – First index in the sequence. (input)
- last (int) – Last index plus 1 in the sequence. (input)
- sux (float[]) – Dual variables corresponding to the upper bounds on the variables. (input)

Groups *Solution - dual*

Task.puttaskname

```
def puttaskname (taskname)
```

Assigns a new name to the task.

Parameters taskname (str) – Name assigned to the task. (input)

Groups *Names*, *Environment and task management*

Task.putvarbound

```
def putvarbound (j, bkx, blx, bux)
```

Changes the bounds for one variable.

If the bound value specified is numerically larger than *dparam.data_tol_bound_inf* it is considered infinite and the bound key is changed accordingly. If a bound value is numerically larger than *dparam.data_tol_bound_wrn*, a warning will be displayed, but the bound is inputted as specified.

Parameters

- j (int) – Index of the variable. (input)
- bkx (*mosek.boundkey*) – New bound key. (input)
- blx (float) – New lower bound. (input)
- bux (float) – New upper bound. (input)

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

Task.putvarboundlist

```
def putvarboundlist (sub, bkx, blx, bux)
```

Changes the bounds for one or more variables. If multiple bound changes are specified for a variable, then only the last change takes effect. Data checks are performed as in *Task.putvarbound*.

Parameters

- sub (int[]) – List of variable indexes. (input)
- bkx (*mosek.boundkey* []) – Bound keys for the variables. (input)
- blx (float[]) – Lower bounds for the variables. (input)
- bux (float[]) – Upper bounds for the variables. (input)

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

Task.putvarboundlistconst

```
def putvarboundlistconst (sub, bkx, blx, bux)
```

Changes the bounds for one or more variables. Data checks are performed as in *Task.putvarbound*.

Parameters

- sub (int[]) – List of variable indexes. (input)
- bkx (*mosek.boundkey*) – New bound key for all variables in the list. (input)
- blx (float) – New lower bound for all variables in the list. (input)
- bux (float) – New upper bound for all variables in the list. (input)

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

Task.putvarboundslice

```
def putvarboundslice (first, last, bkx, blx, bux)
```

Changes the bounds for a slice of the variables. Data checks are performed as in *Task.putvarbound*.

Parameters

- first (int) – First index in the sequence. (input)
- last (int) – Last index plus 1 in the sequence. (input)
- bkx (*mosek.boundkey* []) – Bound keys for the variables. (input)
- blx (float[]) – Lower bounds for the variables. (input)
- bux (float[]) – Upper bounds for the variables. (input)

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

Task.putvarboundsliceconst

```
def putvarboundsliceconst (first, last, bkx, blx, bux)
```

Changes the bounds for a slice of the variables. Data checks are performed as in *Task.putvarbound*.

Parameters

- first (int) – First index in the sequence. (input)
- last (int) – Last index plus 1 in the sequence. (input)
- bkx (*mosek.boundkey*) – New bound key for all variables in the slice. (input)
- blx (float) – New lower bound for all variables in the slice. (input)
- bux (float) – New upper bound for all variables in the slice. (input)

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

Task.putvarname

```
def putvarname (j, name)
```

Sets the name of a variable.

Parameters

- j (int) – Index of the variable. (input)
- name (str) – The variable name. (input)

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

Task.putvarsolutionj

```
def putvarsolutionj (j, whichsol, sk, x, sl, su, sn)
```

Sets the primal and dual solution information for a single variable.

Parameters
- `j` (`int`) – Index of the variable. (input)
- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `sk` (*mosek.stakey*) – Status key of the variable. (input)
- `x` (`float`) – Primal solution value of the variable. (input)
- `sl` (`float`) – Solution value of the dual variable associated with the lower bound. (input)
- `su` (`float`) – Solution value of the dual variable associated with the upper bound. (input)
- `sn` (`float`) – Solution value of the dual variable associated with the conic constraint. (input)

Groups *Solution information*, *Solution - primal*, *Solution - dual*

Task.putvartype

```
def putvartype (j, vartype)
```

Sets the variable type of one variable.

Parameters
- `j` (`int`) – Index of the variable. (input)
- `vartype` (*mosek.variabletype*) – The new variable type. (input)

Groups *Problem data - variables*

Task.putvartypelist

```
def putvartypelist (subj, vartype)
```

Sets the variable type for one or more variables. If the same index is specified multiple times in `subj` only the last entry takes effect.

Parameters
- `subj` (`int[]`) – A list of variable indexes for which the variable type should be changed. (input)
- `vartype` (*mosek.variabletype*`[]`) – A list of variable types that should be assigned to the variables specified by `subj`. (input)

Groups *Problem data - variables*

Task.putxc

```
def putxc (whichsol, xc)
```

Sets the x^c vector for a solution.

Parameters
- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `xc` (`float[]`) – Primal constraint solution. (output)

Groups *Solution - primal*

Task.putxcslice

```
def putxcslice (whichsol, first, last, xc)
```

Sets a slice of the x^c vector for a solution.

Parameters

- whichsol (*mosek.soltype*) – Selects a solution. (input)
- first (int) – First index in the sequence. (input)
- last (int) – Last index plus 1 in the sequence. (input)
- xc (float[]) – Primal constraint solution. (input)

Groups *Solution - primal*

Task.putxx

```
def putxx (whichsol, xx)
```

Sets the x^x vector for a solution.

Parameters

- whichsol (*mosek.soltype*) – Selects a solution. (input)
- xx (float[]) – Primal variable solution. (input)

Groups *Solution - primal*

Task.putxxslice

```
def putxxslice (whichsol, first, last, xx)
```

Sets a slice of the x^x vector for a solution.

Parameters

- whichsol (*mosek.soltype*) – Selects a solution. (input)
- first (int) – First index in the sequence. (input)
- last (int) – Last index plus 1 in the sequence. (input)
- xx (float[]) – Primal variable solution. (input)

Groups *Solution - primal*

Task.puty

```
def puty (whichsol, y)
```

Sets the y vector for a solution.

Parameters

- whichsol (*mosek.soltype*) – Selects a solution. (input)
- y (float[]) – Vector of dual variables corresponding to the constraints. (input)

Groups *Solution - primal*

Task.putyslice

```
def putyslice (whichsol, first, last, y)
```

Sets a slice of the y vector for a solution.

Parameters

- whichsol (*mosek.soltype*) – Selects a solution. (input)

- `first` (`int`) – First index in the sequence. (input)
- `last` (`int`) – Last index plus 1 in the sequence. (input)
- `y` (`float[]`) – Vector of dual variables corresponding to the constraints. (input)

Groups *Solution - dual*

`Task.readdata`

```
def readdata (filename)
```

Reads an optimization problem and associated data from a file.

Parameters `filename` (`str`) – A valid file name. (input)

Groups *Input/Output*

`Task.readdataformat`

```
def readdataformat (filename, format, compress)
```

Reads an optimization problem and associated data from a file.

Parameters

- `filename` (`str`) – A valid file name. (input)
- `format` (*`mosek.dataformat`*) – File data format. (input)
- `compress` (*`mosek.compresstype`*) – File compression type. (input)

Groups *Input/Output*

`Task.readjsonstring`

```
def readjsonstring (data)
```

Load task data from a JSON string, replacing any data that already exists in the task object. All problem data, parameters and other settings are resorted, but if the string contains solutions, the solution status after loading a file is set to unknown, even if it is optimal or otherwise well-defined.

Parameters `data` (`str`) – Problem data in text format. (input)

Groups *Input/Output*

`Task.readlpstring`

```
def readlpstring (data)
```

Load task data from a string in LP format, replacing any data that already exists in the task object.

Parameters `data` (`str`) – Problem data in text format. (input)

Groups *Input/Output*

`Task.readopfstring`

```
def readopfstring (data)
```

Load task data from a string in OPF format, replacing any data that already exists in the task object.

Parameters `data` (`str`) – Problem data in text format. (input)

Groups *Input/Output*

Task.readparamfile

```
def readparamfile (filename)
```

Reads **MOSEK** parameters from a file. Data is read from the file `filename` if it is a nonempty string. Otherwise data is read from the file specified by *`sparam.param_read_file_name`*.

Parameters `filename` (`str`) – A valid file name. (input)

Groups *Input/Output*

Task.readptfstring

```
def readptfstring (data)
```

Load task data from a PTF string, replacing any data that already exists in the task object. All problem data, parameters and other settings are resorted, but if the string contains solutions, the solution status after loading a file is set to unknown, even if it is optimal or otherwise well-defined.

Parameters `data` (`str`) – Problem data in text format. (input)

Groups *Input/Output*

Task.readsolution

```
def readsolution (whichsol, filename)
```

Reads a solution file and inserts it as a specified solution in the task. Data is read from the file `filename` if it is a nonempty string. Otherwise data is read from one of the files specified by *`sparam.bas_sol_file_name`*, *`sparam.itr_sol_file_name`* or *`sparam.int_sol_file_name`* depending on which solution is chosen.

Parameters

- `whichsol` (*`mosek.soltype`*) – Selects a solution. (input)
- `filename` (`str`) – A valid file name. (input)

Groups *Input/Output*

Task.readsummary

```
def readsummary (whichstream)
```

Prints a short summary of last file that was read.

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

Groups *Input/Output*, *Inspecting the task*

Task.readtask

```
def readtask (filename)
```

Load task data from a file, replacing any data that already exists in the task object. All problem data, parameters and other settings are resorted, but if the file contains solutions, the solution status after loading a file is set to unknown, even if it was optimal or otherwise well-defined when the file was dumped.

See section *The Task Format* for a description of the Task format.

Parameters `filename` (`str`) – A valid file name. (input)

Groups *Input/Output*

`Task.removebarvars`

```
def removebarvars (subset)
```

The function removes a subset of the symmetric matrices from the optimization task. This implies that the remaining symmetric matrices are renumbered.

Parameters `subset` (`int[]`) – Indexes of symmetric matrices which should be removed. (input)

Groups *Problem data - semidefinite*

`Task.removecones`

```
def removecones (subset)
```

Removes a number of conic constraints from the problem. This implies that the remaining conic constraints are renumbered. In general, it is much more efficient to remove a cone with a high index than a low index.

Parameters `subset` (`int[]`) – Indexes of cones which should be removed. (input)

Groups *Problem data - cones*

`Task.removecons`

```
def removecons (subset)
```

The function removes a subset of the constraints from the optimization task. This implies that the remaining constraints are renumbered.

Parameters `subset` (`int[]`) – Indexes of constraints which should be removed. (input)

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

`Task.removevars`

```
def removevars (subset)
```

The function removes a subset of the variables from the optimization task. This implies that the remaining variables are renumbered.

Parameters `subset` (`int[]`) – Indexes of variables which should be removed. (input)

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

`Task.resizetask`

```
def resizetask (maxnumcon, maxnumvar, maxnumcone, maxnumanz, maxnumqnz)
```

Sets the amount of preallocated space assigned for each type of data in an optimization task.

It is never mandatory to call this function, since it only gives a hint about the amount of data to preallocate for efficiency reasons.

Please note that the procedure is **destructive** in the sense that all existing data stored in the task is destroyed.

Parameters

- `maxnumcon` (`int`) – New maximum number of constraints. (input)
- `maxnumvar` (`int`) – New maximum number of variables. (input)
- `maxnumcone` (`int`) – New maximum number of cones. (input)
- `maxnumanz` (`int`) – New maximum number of non-zeros in A. (input)

- maxnumqnz (int) – New maximum number of non-zeros in all Q matrices. (input)

Groups *Environment and task management*

Task.sensitivityreport

```
def sensitivityreport (whichstream)
```

Reads a sensitivity format file from a location given by *sparam.sensitivity_file_name* and writes the result to the stream whichstream. If *sparam.sensitivity_res_file_name* is set to a non-empty string, then the sensitivity report is also written to a file of this name.

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

Groups *Sensitivity analysis*

Task.set_InfoCallback

```
def set_InfoCallback (callback)
```

Receive callbacks with solver status and information during optimization.

For example:

```
task.set_InfoCallback(lambda code,dinf,iinf,liinf: print("Called from: {0}".format(code)))
```

Parameters callback (*callbackfunc*) – The callback function. (input)

Task.set_Progress

```
def set_Progress (callback)
```

Receive callbacks about current status of the solver during optimization.

For example:

```
task.set_Progress(lambda code: print("Called from: {0}".format(code)))
```

Parameters callback (*progresscallbackfunc*) – The callback function. (input)

Task.set_Stream

```
def set_Stream (whichstream, callback)
```

Directs all output from a task stream to a callback function.

Parameters

- whichstream (*streamtype*) – Index of the stream. (input)
- callback (*streamfunc*) – The callback function. (input)

Task.setdefaults

```
def setdefaults ()
```

Resets all the parameters to their default values.

Groups *Parameters*

Task.solutiondef

```
def solutiondef (whichsol) -> isdef
```

Checks whether a solution is defined.

Parameters whichsol (*mosek.soltype*) – Selects a solution. (input)
Return isdef (int) – Is non-zero if the requested solution is defined.
Groups *Solution information*

Task.solutionsummary

```
def solutionsummary (whichstream)
```

Prints a short summary of the current solutions.

Parameters whichstream (*mosek.streamtype*) – Index of the stream. (input)
Groups *Logging*, *Solution information*

Task.solvewithbasis

```
def solvewithbasis (transp, numnz, sub, val) -> numnz
```

If a basic solution is available, then exactly *numcon* basis variables are defined. These *numcon* basis variables are denoted the basis. Associated with the basis is a basis matrix denoted B. This function solves either the linear equation system

$$B\overline{X} = b \tag{15.3}$$

or the system

$$B^T\overline{X} = b \tag{15.4}$$

for the unknowns $\overline{X}$, with b being a user-defined vector. In order to make sense of the solution $\overline{X}$ it is important to know the ordering of the variables in the basis because the ordering specifies how B is constructed. When calling *Task.initbasissolve* an ordering of the basis variables is obtained, which can be used to deduce how **MOSEK** has constructed B. Indeed if the k-th basis variable is variable x_j it implies that

$$B_{i,k} = A_{i,j}, \ i = 0, \ldots, numcon - 1.$$

Otherwise if the k-th basis variable is variable x_j^c it implies that

$$B_{i,k} = \begin{cases} -1, & i = j, \\ 0, & i \neq j. \end{cases}$$

The function *Task.initbasissolve* must be called before a call to this function. Please note that this function exploits the sparsity in the vector b to speed up the computations.

Parameters

- transp (int) – If this argument is zero, then (15.3) is solved, if non-zero then (15.4) is solved. (input)
- numnz (int) – As input it is the number of non-zeros in b. As output it is the number of non-zeros in $\overline{X}$. (input/output)
- sub (int[]) – As input it contains the positions of non-zeros in b. As output it contains the positions of the non-zeros in $\overline{X}$. It must have room for *numcon* elements. (input/output)
- val (float[]) – As input it is the vector b as a dense vector (although the positions of non-zeros are specified in sub it is required that val$[i] = 0$ when $b[i] = 0$). As output val is the vector $\overline{X}$ as a dense vector. It must have length *numcon*. (input/output)

Return `numnz` (`int`) – As input it is the number of non-zeros in b. As output it is the number of non-zeros in $\overline{X}$.

Groups *Solving systems with basis matrix*

`Task.strtoconetype`

```
def strtoconetype (str) -> conetype
```

Obtains cone type code corresponding to a cone type string.

Parameters `str` (`str`) – String corresponding to the cone type code `conetype`. (input)

Return `conetype` (*`mosek.conetype`*) – The cone type corresponding to the string `str`.

Groups *Names*

`Task.strtosk`

```
def strtosk (str) -> sk
```

Obtains the status key corresponding to an abbreviation string.

Parameters `str` (`str`) – A status key abbreviation string. (input)

Return `sk` (*`mosek.stakey`*) – Status key corresponding to the string.

Groups *Names*

`Task.toconic`

```
def toconic ()
```

This function tries to reformulate a given Quadratically Constrained Quadratic Optimization problem (QCQP) as a Conic Quadratic Optimization problem (CQO). The first step of the reformulation is to convert the quadratic term of the objective function, if any, into a constraint. Then the following steps are repeated for each quadratic constraint:

- a conic constraint is added along with a suitable number of auxiliary variables and constraints;
- the original quadratic constraint is not removed, but all its coefficients are zeroed out.

Note that the reformulation preserves all the original variables.

The conversion is performed in-place, i.e. the task passed as argument is modified on exit. That also means that if the reformulation fails, i.e. the given QCQP is not representable as a CQO, then the task has an undefined state. In some cases, users may want to clone the task to ensure a clean copy is preserved.

Groups *Problem data - quadratic part*

`Task.updatesolutioninfo`

```
def updatesolutioninfo (whichsol)
```

Update the information items related to the solution.

Parameters `whichsol` (*`mosek.soltype`*) – Selects a solution. (input)

Groups *Information items and statistics*

`Task.writedata`

```
def writedata (filename)
```

Writes problem data associated with the optimization task to a file in one of the supported formats. See Section *Supported File Formats* for the complete list.

The data file format is determined by the file name extension. To write in compressed format append the extension `.gz`. E.g to write a gzip compressed MPS file use the extension `mps.gz`.

Please note that MPS, LP and OPF files require all variables to have unique names. If a task contains no names, it is possible to write the file with automatically generated anonymous names by setting the *iparam.write_generic_names* parameter to *onoffkey.on*.

Data is written to the file `filename` if it is a nonempty string. Otherwise data is written to the file specified by *sparam.data_file_name*.

Parameters `filename` (`str`) – A valid file name. (input)

Groups *Input/Output*

`Task.writejsonsol`

```
def writejsonsol (filename)
```

Saves the current solutions and solver information items in a JSON file.

Parameters `filename` (`str`) – A valid file name. (input)

Groups *Input/Output*

`Task.writeparamfile`

```
def writeparamfile (filename)
```

Writes all the parameters to a parameter file.

Parameters `filename` (`str`) – A valid file name. (input)

Groups *Input/Output*, *Parameters*

`Task.writesolution`

```
def writesolution (whichsol, filename)
```

Saves the current basic, interior-point, or integer solution to a file.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `filename` (`str`) – A valid file name. (input)

Groups *Input/Output*

`Task.writetask`

```
def writetask (filename)
```

Write a binary dump of the task data. This format saves all problem data, coefficients and parameter settings. See section *The Task Format* for a description of the Task format.

Parameters `filename` (`str`) – A valid file name. (input)

Groups *Input/Output*

15.5 Exceptions

MosekException

Base exception class for all **MOSEK** exceptions.

Error

Exception class used for all error response codes from **MOSEK**.

Implements *MosekException*

15.6 Parameters grouped by topic

Analysis

- *dparam.ana_sol_infeas_tol*
- *iparam.ana_sol_basis*
- *iparam.ana_sol_print_violated*
- *iparam.log_ana_pro*

Basis identification

- *dparam.sim_lu_tol_rel_piv*
- *iparam.bi_clean_optimizer*
- *iparam.bi_ignore_max_iter*
- *iparam.bi_ignore_num_error*
- *iparam.bi_max_iterations*
- *iparam.intpnt_basis*
- *iparam.log_bi*
- *iparam.log_bi_freq*

Conic interior-point method

- *dparam.intpnt_co_tol_dfeas*
- *dparam.intpnt_co_tol_infeas*
- *dparam.intpnt_co_tol_mu_red*
- *dparam.intpnt_co_tol_near_rel*
- *dparam.intpnt_co_tol_pfeas*
- *dparam.intpnt_co_tol_rel_gap*

Data check

- *dparam.data_sym_mat_tol*
- *dparam.data_sym_mat_tol_huge*
- *dparam.data_sym_mat_tol_large*
- *dparam.data_tol_aij_huge*
- *dparam.data_tol_aij_large*
- *dparam.data_tol_bound_inf*
- *dparam.data_tol_bound_wrn*
- *dparam.data_tol_c_huge*
- *dparam.data_tol_cj_large*
- *dparam.data_tol_qij*
- *dparam.data_tol_x*
- *dparam.semidefinite_tol_approx*
- *iparam.check_convexity*
- *iparam.log_check_convexity*

Data input/output

- *iparam.infeas_report_auto*
- *iparam.log_file*
- *iparam.opf_write_header*
- *iparam.opf_write_hints*
- *iparam.opf_write_line_length*
- *iparam.opf_write_parameters*
- *iparam.opf_write_problem*
- *iparam.opf_write_sol_bas*
- *iparam.opf_write_sol_itg*
- *iparam.opf_write_sol_itr*
- *iparam.opf_write_solutions*
- *iparam.param_read_case_name*
- *iparam.param_read_ign_error*
- *iparam.ptf_write_transform*
- *iparam.read_debug*
- *iparam.read_keep_free_con*
- *iparam.read_lp_drop_new_vars_in_bou*
- *iparam.read_lp_quoted_names*
- *iparam.read_mps_format*

- *iparam.read_mps_width*
- *iparam.read_task_ignore_param*
- *iparam.sol_read_name_width*
- *iparam.sol_read_width*
- *iparam.write_bas_constraints*
- *iparam.write_bas_head*
- *iparam.write_bas_variables*
- *iparam.write_compression*
- *iparam.write_data_param*
- *iparam.write_free_con*
- *iparam.write_generic_names*
- *iparam.write_generic_names_io*
- *iparam.write_ignore_incompatible_items*
- *iparam.write_int_constraints*
- *iparam.write_int_head*
- *iparam.write_int_variables*
- *iparam.write_lp_full_obj*
- *iparam.write_lp_line_width*
- *iparam.write_lp_quoted_names*
- *iparam.write_lp_strict_format*
- *iparam.write_lp_terms_per_line*
- *iparam.write_mps_format*
- *iparam.write_mps_int*
- *iparam.write_precision*
- *iparam.write_sol_barvariables*
- *iparam.write_sol_constraints*
- *iparam.write_sol_head*
- *iparam.write_sol_ignore_invalid_names*
- *iparam.write_sol_variables*
- *iparam.write_task_inc_sol*
- *iparam.write_xml_mode*
- *sparam.bas_sol_file_name*
- *sparam.data_file_name*
- *sparam.debug_file_name*
- *sparam.int_sol_file_name*
- *sparam.itr_sol_file_name*

- *sparam.mio_debug_string*
- *sparam.param_comment_sign*
- *sparam.param_read_file_name*
- *sparam.param_write_file_name*
- *sparam.read_mps_bou_name*
- *sparam.read_mps_obj_name*
- *sparam.read_mps_ran_name*
- *sparam.read_mps_rhs_name*
- *sparam.sensitivity_file_name*
- *sparam.sensitivity_res_file_name*
- *sparam.sol_filter_xc_low*
- *sparam.sol_filter_xc_upr*
- *sparam.sol_filter_xx_low*
- *sparam.sol_filter_xx_upr*
- *sparam.stat_file_name*
- *sparam.stat_key*
- *sparam.stat_name*
- *sparam.write_lp_gen_var_name*

Debugging

- *iparam.auto_sort_a_before_opt*

Dual simplex

- *iparam.sim_dual_crash*
- *iparam.sim_dual_restrict_selection*
- *iparam.sim_dual_selection*

Infeasibility report

- *iparam.infeas_generic_names*
- *iparam.infeas_report_level*
- *iparam.log_infeas_ana*

Interior-point method

- *dparam.check_convexity_rel_tol*
- *dparam.intpnt_co_tol_dfeas*
- *dparam.intpnt_co_tol_infeas*
- *dparam.intpnt_co_tol_mu_red*
- *dparam.intpnt_co_tol_near_rel*
- *dparam.intpnt_co_tol_pfeas*
- *dparam.intpnt_co_tol_rel_gap*
- *dparam.intpnt_qo_tol_dfeas*
- *dparam.intpnt_qo_tol_infeas*
- *dparam.intpnt_qo_tol_mu_red*
- *dparam.intpnt_qo_tol_near_rel*
- *dparam.intpnt_qo_tol_pfeas*
- *dparam.intpnt_qo_tol_rel_gap*
- *dparam.intpnt_tol_dfeas*
- *dparam.intpnt_tol_dsafe*
- *dparam.intpnt_tol_infeas*
- *dparam.intpnt_tol_mu_red*
- *dparam.intpnt_tol_path*
- *dparam.intpnt_tol_pfeas*
- *dparam.intpnt_tol_psafe*
- *dparam.intpnt_tol_rel_gap*
- *dparam.intpnt_tol_rel_step*
- *dparam.intpnt_tol_step_size*
- *dparam.qcqo_reformulate_rel_drop_tol*
- *iparam.bi_ignore_max_iter*
- *iparam.bi_ignore_num_error*
- *iparam.intpnt_basis*
- *iparam.intpnt_diff_step*
- *iparam.intpnt_hotstart*
- *iparam.intpnt_max_iterations*
- *iparam.intpnt_max_num_cor*
- *iparam.intpnt_max_num_refinement_steps*
- *iparam.intpnt_off_col_trh*
- *iparam.intpnt_order_gp_num_seeds*
- *iparam.intpnt_order_method*

- *iparam.intpnt_purify*
- *iparam.intpnt_regularization_use*
- *iparam.intpnt_scaling*
- *iparam.intpnt_solve_form*
- *iparam.intpnt_starting_point*
- *iparam.log_intpnt*

License manager

- *iparam.cache_license*
- *iparam.license_debug*
- *iparam.license_pause_time*
- *iparam.license_suppress_expire_wrns*
- *iparam.license_trh_expiry_wrn*
- *iparam.license_wait*

Logging

- *iparam.log*
- *iparam.log_ana_pro*
- *iparam.log_bi*
- *iparam.log_bi_freq*
- *iparam.log_cut_second_opt*
- *iparam.log_expand*
- *iparam.log_feas_repair*
- *iparam.log_file*
- *iparam.log_include_summary*
- *iparam.log_infeas_ana*
- *iparam.log_intpnt*
- *iparam.log_local_info*
- *iparam.log_mio*
- *iparam.log_mio_freq*
- *iparam.log_order*
- *iparam.log_presolve*
- *iparam.log_response*
- *iparam.log_sensitivity*
- *iparam.log_sensitivity_opt*
- *iparam.log_sim*
- *iparam.log_sim_freq*
- *iparam.log_storage*

Mixed-integer optimization

- *dparam.mio_max_time*
- *dparam.mio_rel_gap_const*
- *dparam.mio_tol_abs_gap*
- *dparam.mio_tol_abs_relax_int*
- *dparam.mio_tol_feas*
- *dparam.mio_tol_rel_dual_bound_improvement*
- *dparam.mio_tol_rel_gap*
- *iparam.log_mio*
- *iparam.log_mio_freq*
- *iparam.mio_branch_dir*
- *iparam.mio_conic_outer_approximation*
- *iparam.mio_cut_clique*
- *iparam.mio_cut_cmir*
- *iparam.mio_cut_gmi*
- *iparam.mio_cut_implied_bound*
- *iparam.mio_cut_knapsack_cover*
- *iparam.mio_cut_selection_level*
- *iparam.mio_feaspump_level*
- *iparam.mio_heuristic_level*
- *iparam.mio_max_num_branches*
- *iparam.mio_max_num_relaxs*
- *iparam.mio_max_num_root_cut_rounds*
- *iparam.mio_max_num_solutions*
- *iparam.mio_node_optimizer*
- *iparam.mio_node_selection*
- *iparam.mio_perspective_reformulate*
- *iparam.mio_probing_level*
- *iparam.mio_propagate_objective_constraint*
- *iparam.mio_rins_max_nodes*
- *iparam.mio_root_optimizer*
- *iparam.mio_root_repeat_presolve_level*
- *iparam.mio_seed*
- *iparam.mio_vb_detection_level*

Output information

- *iparam.infeas_report_level*
- *iparam.license_suppress_expire_wrns*
- *iparam.license_trh_expiry_wrn*
- *iparam.log*
- *iparam.log_bi*
- *iparam.log_bi_freq*
- *iparam.log_cut_second_opt*
- *iparam.log_expand*
- *iparam.log_feas_repair*
- *iparam.log_file*
- *iparam.log_include_summary*
- *iparam.log_infeas_ana*
- *iparam.log_intpnt*
- *iparam.log_local_info*
- *iparam.log_mio*
- *iparam.log_mio_freq*
- *iparam.log_order*
- *iparam.log_response*
- *iparam.log_sensitivity*
- *iparam.log_sensitivity_opt*
- *iparam.log_sim*
- *iparam.log_sim_freq*
- *iparam.log_sim_minor*
- *iparam.log_storage*
- *iparam.max_num_warnings*

Overall solver

- *iparam.bi_clean_optimizer*
- *iparam.infeas_prefer_primal*
- *iparam.license_wait*
- *iparam.mio_mode*
- *iparam.optimizer*
- *iparam.presolve_level*
- *iparam.presolve_max_num_reductions*
- *iparam.presolve_use*

- *iparam.primal_repair_optimizer*
- *iparam.sensitivity_all*
- *iparam.sensitivity_optimizer*
- *iparam.sensitivity_type*
- *iparam.solution_callback*

Overall system

- *iparam.auto_update_sol_info*
- *iparam.intpnt_multi_thread*
- *iparam.license_wait*
- *iparam.log_storage*
- *iparam.mt_spincount*
- *iparam.num_threads*
- *iparam.remove_unused_solutions*
- *iparam.timing_level*
- *sparam.remote_access_token*

Presolve

- *dparam.presolve_tol_abs_lindep*
- *dparam.presolve_tol_aij*
- *dparam.presolve_tol_rel_lindep*
- *dparam.presolve_tol_s*
- *dparam.presolve_tol_x*
- *iparam.presolve_eliminator_max_fill*
- *iparam.presolve_eliminator_max_num_tries*
- *iparam.presolve_level*
- *iparam.presolve_lindep_abs_work_trh*
- *iparam.presolve_lindep_rel_work_trh*
- *iparam.presolve_lindep_use*
- *iparam.presolve_max_num_pass*
- *iparam.presolve_max_num_reductions*
- *iparam.presolve_use*

Primal simplex

- *iparam.sim_primal_crash*
- *iparam.sim_primal_restrict_selection*
- *iparam.sim_primal_selection*

Progress callback

- *iparam.solution_callback*

Simplex optimizer

- *dparam.basis_rel_tol_s*
- *dparam.basis_tol_s*
- *dparam.basis_tol_x*
- *dparam.sim_lu_tol_rel_piv*
- *dparam.simplex_abs_tol_piv*
- *iparam.basis_solve_use_plus_one*
- *iparam.log_sim*
- *iparam.log_sim_freq*
- *iparam.log_sim_minor*
- *iparam.sensitivity_optimizer*
- *iparam.sim_basis_factor_use*
- *iparam.sim_degen*
- *iparam.sim_dual_phaseone_method*
- *iparam.sim_exploit_dupvec*
- *iparam.sim_hotstart*
- *iparam.sim_hotstart_lu*
- *iparam.sim_max_iterations*
- *iparam.sim_max_num_setbacks*
- *iparam.sim_non_singular*
- *iparam.sim_primal_phaseone_method*
- *iparam.sim_refactor_freq*
- *iparam.sim_reformulation*
- *iparam.sim_save_lu*
- *iparam.sim_scaling*
- *iparam.sim_scaling_method*
- *iparam.sim_seed*
- *iparam.sim_solve_form*
- *iparam.sim_stability_priority*
- *iparam.sim_switch_optimizer*

Solution input/output

- *iparam.infeas_report_auto*
- *iparam.sol_filter_keep_basic*
- *iparam.sol_filter_keep_ranged*
- *iparam.sol_read_name_width*
- *iparam.sol_read_width*
- *iparam.write_bas_constraints*
- *iparam.write_bas_head*
- *iparam.write_bas_variables*
- *iparam.write_int_constraints*
- *iparam.write_int_head*
- *iparam.write_int_variables*
- *iparam.write_sol_barvariables*
- *iparam.write_sol_constraints*
- *iparam.write_sol_head*
- *iparam.write_sol_ignore_invalid_names*
- *iparam.write_sol_variables*
- *sparam.bas_sol_file_name*
- *sparam.int_sol_file_name*
- *sparam.itr_sol_file_name*
- *sparam.sol_filter_xc_low*
- *sparam.sol_filter_xc_upr*
- *sparam.sol_filter_xx_low*
- *sparam.sol_filter_xx_upr*

Termination criteria

- *dparam.basis_rel_tol_s*
- *dparam.basis_tol_s*
- *dparam.basis_tol_x*
- *dparam.intpnt_co_tol_dfeas*
- *dparam.intpnt_co_tol_infeas*
- *dparam.intpnt_co_tol_mu_red*
- *dparam.intpnt_co_tol_near_rel*
- *dparam.intpnt_co_tol_pfeas*
- *dparam.intpnt_co_tol_rel_gap*
- *dparam.intpnt_qo_tol_dfeas*

- *dparam.intpnt_qo_tol_infeas*
- *dparam.intpnt_qo_tol_mu_red*
- *dparam.intpnt_qo_tol_near_rel*
- *dparam.intpnt_qo_tol_pfeas*
- *dparam.intpnt_qo_tol_rel_gap*
- *dparam.intpnt_tol_dfeas*
- *dparam.intpnt_tol_infeas*
- *dparam.intpnt_tol_mu_red*
- *dparam.intpnt_tol_pfeas*
- *dparam.intpnt_tol_rel_gap*
- *dparam.lower_obj_cut*
- *dparam.lower_obj_cut_finite_trh*
- *dparam.mio_max_time*
- *dparam.mio_rel_gap_const*
- *dparam.mio_tol_rel_gap*
- *dparam.optimizer_max_time*
- *dparam.upper_obj_cut*
- *dparam.upper_obj_cut_finite_trh*
- *iparam.bi_max_iterations*
- *iparam.intpnt_max_iterations*
- *iparam.mio_max_num_branches*
- *iparam.mio_max_num_root_cut_rounds*
- *iparam.mio_max_num_solutions*
- *iparam.sim_max_iterations*

Other

- *iparam.compress_statfile*

15.7 Parameters (alphabetical list sorted by type)

- *Double parameters*
- *Integer parameters*
- *String parameters*

15.7.1 Double parameters

`dparam`

The enumeration type containing all double parameters.

`dparam.ana_sol_infeas_tol`

If a constraint violates its bound with an amount larger than this value, the constraint name, index and violation will be printed by the solution analyzer.

Default 1e-6

Accepted [0.0; +inf]

Example `task.putdouparam(dparam.ana_sol_infeas_tol, 1e-6)`

Generic name MSK_DPAR_ANA_SOL_INFEAS_TOL

Groups *Analysis*

`dparam.basis_rel_tol_s`

Maximum relative dual bound violation allowed in an optimal basic solution.

Default 1.0e-12

Accepted [0.0; +inf]

Example `task.putdouparam(dparam.basis_rel_tol_s, 1.0e-12)`

Generic name MSK_DPAR_BASIS_REL_TOL_S

Groups *Simplex optimizer*, *Termination criteria*

`dparam.basis_tol_s`

Maximum absolute dual bound violation in an optimal basic solution.

Default 1.0e-6

Accepted [1.0e-9; +inf]

Example `task.putdouparam(dparam.basis_tol_s, 1.0e-6)`

Generic name MSK_DPAR_BASIS_TOL_S

Groups *Simplex optimizer*, *Termination criteria*

`dparam.basis_tol_x`

Maximum absolute primal bound violation allowed in an optimal basic solution.

Default 1.0e-6

Accepted [1.0e-9; +inf]

Example `task.putdouparam(dparam.basis_tol_x, 1.0e-6)`

Generic name MSK_DPAR_BASIS_TOL_X

Groups *Simplex optimizer*, *Termination criteria*

`dparam.check_convexity_rel_tol`

This parameter controls when the full convexity check declares a problem to be non-convex. Increasing this tolerance relaxes the criteria for declaring the problem non-convex.

A problem is declared non-convex if negative (positive) pivot elements are detected in the Cholesky factor of a matrix which is required to be PSD (NSD). This parameter controls how much this non-negativity requirement may be violated.

If d_i is the pivot element for column i, then the matrix Q is considered to not be PSD if:

$$d_i \leq -|Q_{ii}|\texttt{check\_convexity\_rel\_tol}$$

Default 1e-10

Accepted [0; +inf]

Example `task.putdouparam(dparam.check_convexity_rel_tol, 1e-10)`

Generic name MSK_DPAR_CHECK_CONVEXITY_REL_TOL

Groups *Interior-point method*

`dparam.data_sym_mat_tol`

Absolute zero tolerance for elements in in symmetric matrices. If any value in a symmetric matrix is smaller than this parameter in absolute terms **MOSEK** will treat the values as zero and generate a warning.

Default 1.0e-12

Accepted [1.0e-16; 1.0e-6]

Example `task.putdouparam(dparam.data_sym_mat_tol, 1.0e-12)`

Generic name MSK_DPAR_DATA_SYM_MAT_TOL

Groups *Data check*

`dparam.data_sym_mat_tol_huge`

An element in a symmetric matrix which is larger than this value in absolute size causes an error.

Default 1.0e20

Accepted [0.0; +inf]

Example `task.putdouparam(dparam.data_sym_mat_tol_huge, 1.0e20)`

Generic name MSK_DPAR_DATA_SYM_MAT_TOL_HUGE

Groups *Data check*

`dparam.data_sym_mat_tol_large`

An element in a symmetric matrix which is larger than this value in absolute size causes a warning message to be printed.

Default 1.0e10

Accepted [0.0; +inf]

Example `task.putdouparam(dparam.data_sym_mat_tol_large, 1.0e10)`

Generic name MSK_DPAR_DATA_SYM_MAT_TOL_LARGE

Groups *Data check*

`dparam.data_tol_aij_huge`

An element in A which is larger than this value in absolute size causes an error.

Default 1.0e20

Accepted [0.0; +inf]

Example `task.putdouparam(dparam.data_tol_aij_huge, 1.0e20)`

Generic name MSK_DPAR_DATA_TOL_AIJ_HUGE

Groups *Data check*

`dparam.data_tol_aij_large`

An element in A which is larger than this value in absolute size causes a warning message to be printed.

Default 1.0e10

Accepted [0.0; +inf]

Example `task.putdouparam(dparam.data_tol_aij_large, 1.0e10)`

Generic name MSK_DPAR_DATA_TOL_AIJ_LARGE

Groups *Data check*

`dparam.data_tol_bound_inf`

Any bound which in absolute value is greater than this parameter is considered infinite.

Default 1.0e16

Accepted [0.0; +inf]

Example `task.putdouparam(dparam.data_tol_bound_inf, 1.0e16)`

Generic name MSK_DPAR_DATA_TOL_BOUND_INF

Groups *Data check*

`dparam.data_tol_bound_wrn`

If a bound value is larger than this value in absolute size, then a warning message is issued.

Default 1.0e8

Accepted [0.0; +inf]

Example `task.putdouparam(dparam.data_tol_bound_wrn, 1.0e8)`

Generic name MSK_DPAR_DATA_TOL_BOUND_WRN

Groups *Data check*

`dparam.data_tol_c_huge`

An element in c which is larger than the value of this parameter in absolute terms is considered to be huge and generates an error.

Default 1.0e16

Accepted [0.0; +inf]

Example `task.putdouparam(dparam.data_tol_c_huge, 1.0e16)`

Generic name MSK_DPAR_DATA_TOL_C_HUGE

Groups *Data check*

`dparam.data_tol_cj_large`

An element in c which is larger than this value in absolute terms causes a warning message to be printed.

Default 1.0e8

Accepted [0.0; +inf]

Example `task.putdouparam(dparam.data_tol_cj_large, 1.0e8)`

Generic name MSK_DPAR_DATA_TOL_CJ_LARGE

Groups *Data check*

`dparam.data_tol_qij`

Absolute zero tolerance for elements in Q matrices.

Default 1.0e-16

Accepted [0.0; +inf]

Example `task.putdouparam(dparam.data_tol_qij, 1.0e-16)`

Generic name MSK_DPAR_DATA_TOL_QIJ

Groups *Data check*

`dparam.data_tol_x`

Zero tolerance for constraints and variables i.e. if the distance between the lower and upper bound is less than this value, then the lower and upper bound is considered identical.

Default 1.0e-8

Accepted [0.0; +inf]

Example `task.putdouparam(dparam.data_tol_x, 1.0e-8)`

Generic name MSK_DPAR_DATA_TOL_X

Groups *Data check*

`dparam.intpnt_co_tol_dfeas`

Dual feasibility tolerance used by the interior-point optimizer for conic problems.

Default 1.0e-8

Accepted [0.0; 1.0]

Example `task.putdouparam(dparam.intpnt_co_tol_dfeas, 1.0e-8)`

See also *dparam.intpnt_co_tol_near_rel*

Generic name MSK_DPAR_INTPNT_CO_TOL_DFEAS

Groups *Interior-point method*, *Termination criteria*, *Conic interior-point method*

dparam.intpnt_co_tol_infeas
Infeasibility tolerance used by the interior-point optimizer for conic problems. Controls when the interior-point optimizer declares the model primal or dual infeasible. A small number means the optimizer gets more conservative about declaring the model infeasible.

Default 1.0e-12

Accepted [0.0; 1.0]

Example `task.putdouparam(dparam.intpnt_co_tol_infeas, 1.0e-12)`

Generic name MSK_DPAR_INTPNT_CO_TOL_INFEAS

Groups *Interior-point method*, *Termination criteria*, *Conic interior-point method*

dparam.intpnt_co_tol_mu_red
Relative complementarity gap tolerance used by the interior-point optimizer for conic problems.

Default 1.0e-8

Accepted [0.0; 1.0]

Example `task.putdouparam(dparam.intpnt_co_tol_mu_red, 1.0e-8)`

Generic name MSK_DPAR_INTPNT_CO_TOL_MU_RED

Groups *Interior-point method*, *Termination criteria*, *Conic interior-point method*

dparam.intpnt_co_tol_near_rel
Optimality tolerance used by the interior-point optimizer for conic problems. If **MOSEK** cannot compute a solution that has the prescribed accuracy then it will check if the solution found satisfies the termination criteria with all tolerances multiplied by the value of this parameter. If yes, then the solution is also declared optimal.

Default 1000

Accepted [1.0; +inf]

Example `task.putdouparam(dparam.intpnt_co_tol_near_rel, 1000)`

Generic name MSK_DPAR_INTPNT_CO_TOL_NEAR_REL

Groups *Interior-point method*, *Termination criteria*, *Conic interior-point method*

dparam.intpnt_co_tol_pfeas
Primal feasibility tolerance used by the interior-point optimizer for conic problems.

Default 1.0e-8

Accepted [0.0; 1.0]

Example `task.putdouparam(dparam.intpnt_co_tol_pfeas, 1.0e-8)`

See also *dparam.intpnt_co_tol_near_rel*

Generic name MSK_DPAR_INTPNT_CO_TOL_PFEAS

Groups *Interior-point method*, *Termination criteria*, *Conic interior-point method*

dparam.intpnt_co_tol_rel_gap
Relative gap termination tolerance used by the interior-point optimizer for conic problems.

Default 1.0e-8

Accepted [0.0; 1.0]

Example `task.putdouparam(dparam.intpnt_co_tol_rel_gap, 1.0e-8)`

See also *dparam.intpnt_co_tol_near_rel*

Generic name MSK_DPAR_INTPNT_CO_TOL_REL_GAP

Groups *Interior-point method*, *Termination criteria*, *Conic interior-point method*

dparam.intpnt_qo_tol_dfeas
Dual feasibility tolerance used by the interior-point optimizer for quadratic problems.

Default 1.0e-8

Accepted [0.0; 1.0]

Example `task.putdouparam(dparam.intpnt_qo_tol_dfeas, 1.0e-8)`

See also *dparam.intpnt_qo_tol_near_rel*

Generic name MSK_DPAR_INTPNT_QO_TOL_DFEAS

Groups *Interior-point method*, *Termination criteria*

`dparam.intpnt_qo_tol_infeas`

Infeasibility tolerance used by the interior-point optimizer for quadratic problems. Controls when the interior-point optimizer declares the model primal or dual infeasible. A small number means the optimizer gets more conservative about declaring the model infeasible.

Default 1.0e-12

Accepted [0.0; 1.0]

Example `task.putdouparam(dparam.intpnt_qo_tol_infeas, 1.0e-12)`

Generic name MSK_DPAR_INTPNT_QO_TOL_INFEAS

Groups *Interior-point method*, *Termination criteria*

`dparam.intpnt_qo_tol_mu_red`

Relative complementarity gap tolerance used by the interior-point optimizer for quadratic problems.

Default 1.0e-8

Accepted [0.0; 1.0]

Example `task.putdouparam(dparam.intpnt_qo_tol_mu_red, 1.0e-8)`

Generic name MSK_DPAR_INTPNT_QO_TOL_MU_RED

Groups *Interior-point method*, *Termination criteria*

`dparam.intpnt_qo_tol_near_rel`

Optimality tolerance used by the interior-point optimizer for quadratic problems. If **MOSEK** cannot compute a solution that has the prescribed accuracy then it will check if the solution found satisfies the termination criteria with all tolerances multiplied by the value of this parameter. If yes, then the solution is also declared optimal.

Default 1000

Accepted [1.0; +inf]

Example `task.putdouparam(dparam.intpnt_qo_tol_near_rel, 1000)`

Generic name MSK_DPAR_INTPNT_QO_TOL_NEAR_REL

Groups *Interior-point method*, *Termination criteria*

`dparam.intpnt_qo_tol_pfeas`

Primal feasibility tolerance used by the interior-point optimizer for quadratic problems.

Default 1.0e-8

Accepted [0.0; 1.0]

Example `task.putdouparam(dparam.intpnt_qo_tol_pfeas, 1.0e-8)`

See also *dparam.intpnt_qo_tol_near_rel*

Generic name MSK_DPAR_INTPNT_QO_TOL_PFEAS

Groups *Interior-point method*, *Termination criteria*

`dparam.intpnt_qo_tol_rel_gap`

Relative gap termination tolerance used by the interior-point optimizer for quadratic problems.

Default 1.0e-8

Accepted [0.0; 1.0]

Example `task.putdouparam(dparam.intpnt_qo_tol_rel_gap, 1.0e-8)`

See also *dparam.intpnt_qo_tol_near_rel*

Generic name MSK_DPAR_INTPNT_QO_TOL_REL_GAP

Groups *Interior-point method*, *Termination criteria*

`dparam.intpnt_tol_dfeas`

Dual feasibility tolerance used by the interior-point optimizer for linear problems.

Default 1.0e-8

Accepted [0.0; 1.0]

Example `task.putdouparam(dparam.intpnt_tol_dfeas, 1.0e-8)`

Generic name MSK_DPAR_INTPNT_TOL_DFEAS

Groups *Interior-point method*, *Termination criteria*

`dparam.intpnt_tol_dsafe`

Controls the initial dual starting point used by the interior-point optimizer. If the interior-point optimizer converges slowly and/or the constraint or variable bounds are very large, then it might be worthwhile to increase this value.

Default 1.0

Accepted [1.0e-4; +inf]

Example `task.putdouparam(dparam.intpnt_tol_dsafe, 1.0)`

Generic name MSK_DPAR_INTPNT_TOL_DSAFE

Groups *Interior-point method*

`dparam.intpnt_tol_infeas`

Infeasibility tolerance used by the interior-point optimizer for linear problems. Controls when the interior-point optimizer declares the model primal or dual infeasible. A small number means the optimizer gets more conservative about declaring the model infeasible.

Default 1.0e-10

Accepted [0.0; 1.0]

Example `task.putdouparam(dparam.intpnt_tol_infeas, 1.0e-10)`

Generic name MSK_DPAR_INTPNT_TOL_INFEAS

Groups *Interior-point method*, *Termination criteria*

`dparam.intpnt_tol_mu_red`

Relative complementarity gap tolerance used by the interior-point optimizer for linear problems.

Default 1.0e-16

Accepted [0.0; 1.0]

Example `task.putdouparam(dparam.intpnt_tol_mu_red, 1.0e-16)`

Generic name MSK_DPAR_INTPNT_TOL_MU_RED

Groups *Interior-point method*, *Termination criteria*

`dparam.intpnt_tol_path`

Controls how close the interior-point optimizer follows the central path. A large value of this parameter means the central path is followed very closely. On numerically unstable problems it may be worthwhile to increase this parameter.

Default 1.0e-8

Accepted [0.0; 0.9999]

Example `task.putdouparam(dparam.intpnt_tol_path, 1.0e-8)`

Generic name MSK_DPAR_INTPNT_TOL_PATH

Groups *Interior-point method*

`dparam.intpnt_tol_pfeas`

Primal feasibility tolerance used by the interior-point optimizer for linear problems.

Default 1.0e-8

Accepted [0.0; 1.0]

Example `task.putdouparam(dparam.intpnt_tol_pfeas, 1.0e-8)`

Generic name MSK_DPAR_INTPNT_TOL_PFEAS

Groups *Interior-point method*, *Termination criteria*

`dparam.intpnt_tol_psafe`
Controls the initial primal starting point used by the interior-point optimizer. If the interior-point optimizer converges slowly and/or the constraint or variable bounds are very large, then it may be worthwhile to increase this value.

Default 1.0
Accepted [1.0e-4; +inf]
Example `task.putdouparam(dparam.intpnt_tol_psafe, 1.0)`
Generic name MSK_DPAR_INTPNT_TOL_PSAFE
Groups *Interior-point method*

`dparam.intpnt_tol_rel_gap`
Relative gap termination tolerance used by the interior-point optimizer for linear problems.

Default 1.0e-8
Accepted [1.0e-14; +inf]
Example `task.putdouparam(dparam.intpnt_tol_rel_gap, 1.0e-8)`
Generic name MSK_DPAR_INTPNT_TOL_REL_GAP
Groups *Termination criteria*, *Interior-point method*

`dparam.intpnt_tol_rel_step`
Relative step size to the boundary for linear and quadratic optimization problems.

Default 0.9999
Accepted [1.0e-4; 0.999999]
Example `task.putdouparam(dparam.intpnt_tol_rel_step, 0.9999)`
Generic name MSK_DPAR_INTPNT_TOL_REL_STEP
Groups *Interior-point method*

`dparam.intpnt_tol_step_size`
Minimal step size tolerance. If the step size falls below the value of this parameter, then the interior-point optimizer assumes that it is stalled. In other words the interior-point optimizer does not make any progress and therefore it is better to stop.

Default 1.0e-6
Accepted [0.0; 1.0]
Example `task.putdouparam(dparam.intpnt_tol_step_size, 1.0e-6)`
Generic name MSK_DPAR_INTPNT_TOL_STEP_SIZE
Groups *Interior-point method*

`dparam.lower_obj_cut`
If either a primal or dual feasible solution is found proving that the optimal objective value is outside the interval [*dparam.lower_obj_cut* , *dparam.upper_obj_cut*], then **MOSEK** is terminated.

Default -1.0e30
Accepted [-inf; +inf]
Example `task.putdouparam(dparam.lower_obj_cut, -1.0e30)`
See also *dparam.lower_obj_cut_finite_trh*
Generic name MSK_DPAR_LOWER_OBJ_CUT
Groups *Termination criteria*

`dparam.lower_obj_cut_finite_trh`
If the lower objective cut is less than the value of this parameter value, then the lower objective cut i.e. *dparam.lower_obj_cut* is treated as $-\infty$.

Default -0.5e30
Accepted [-inf; +inf]
Example `task.putdouparam(dparam.lower_obj_cut_finite_trh, -0.5e30)`

Generic name MSK_DPAR_LOWER_OBJ_CUT_FINITE_TRH

Groups *Termination criteria*

`dparam.mio_max_time`

This parameter limits the maximum time spent by the mixed-integer optimizer. A negative number means infinity.

Default -1.0

Accepted [-inf; +inf]

Example `task.putdouparam(dparam.mio_max_time, -1.0)`

Generic name MSK_DPAR_MIO_MAX_TIME

Groups *Mixed-integer optimization*, *Termination criteria*

`dparam.mio_rel_gap_const`

This value is used to compute the relative gap for the solution to an integer optimization problem.

Default 1.0e-10

Accepted [1.0e-15; +inf]

Example `task.putdouparam(dparam.mio_rel_gap_const, 1.0e-10)`

Generic name MSK_DPAR_MIO_REL_GAP_CONST

Groups *Mixed-integer optimization*, *Termination criteria*

`dparam.mio_tol_abs_gap`

Absolute optimality tolerance employed by the mixed-integer optimizer.

Default 0.0

Accepted [0.0; +inf]

Example `task.putdouparam(dparam.mio_tol_abs_gap, 0.0)`

Generic name MSK_DPAR_MIO_TOL_ABS_GAP

Groups *Mixed-integer optimization*

`dparam.mio_tol_abs_relax_int`

Absolute integer feasibility tolerance. If the distance to the nearest integer is less than this tolerance then an integer constraint is assumed to be satisfied.

Default 1.0e-5

Accepted [1e-9; +inf]

Example `task.putdouparam(dparam.mio_tol_abs_relax_int, 1.0e-5)`

Generic name MSK_DPAR_MIO_TOL_ABS_RELAX_INT

Groups *Mixed-integer optimization*

`dparam.mio_tol_feas`

Feasibility tolerance for mixed integer solver.

Default 1.0e-6

Accepted [1e-9; 1e-3]

Example `task.putdouparam(dparam.mio_tol_feas, 1.0e-6)`

Generic name MSK_DPAR_MIO_TOL_FEAS

Groups *Mixed-integer optimization*

`dparam.mio_tol_rel_dual_bound_improvement`

If the relative improvement of the dual bound is smaller than this value, the solver will terminate the root cut generation. A value of 0.0 means that the value is selected automatically.

Default 0.0

Accepted [0.0; 1.0]

Example `task.putdouparam(dparam.mio_tol_rel_dual_bound_improvement, 0.0)`

Generic name MSK_DPAR_MIO_TOL_REL_DUAL_BOUND_IMPROVEMENT

Groups *Mixed-integer optimization*

`dparam.mio_tol_rel_gap`

Relative optimality tolerance employed by the mixed-integer optimizer.

Default 1.0e-4

Accepted [0.0; +inf]

Example `task.putdouparam(dparam.mio_tol_rel_gap, 1.0e-4)`

Generic name MSK_DPAR_MIO_TOL_REL_GAP

Groups *Mixed-integer optimization*, *Termination criteria*

`dparam.optimizer_max_time`

Maximum amount of time the optimizer is allowed to spent on the optimization. A negative number means infinity.

Default -1.0

Accepted [-inf; +inf]

Example `task.putdouparam(dparam.optimizer_max_time, -1.0)`

Generic name MSK_DPAR_OPTIMIZER_MAX_TIME

Groups *Termination criteria*

`dparam.presolve_tol_abs_lindep`

Absolute tolerance employed by the linear dependency checker.

Default 1.0e-6

Accepted [0.0; +inf]

Example `task.putdouparam(dparam.presolve_tol_abs_lindep, 1.0e-6)`

Generic name MSK_DPAR_PRESOLVE_TOL_ABS_LINDEP

Groups *Presolve*

`dparam.presolve_tol_aij`

Absolute zero tolerance employed for a_{ij} in the presolve.

Default 1.0e-12

Accepted [1.0e-15; +inf]

Example `task.putdouparam(dparam.presolve_tol_aij, 1.0e-12)`

Generic name MSK_DPAR_PRESOLVE_TOL_AIJ

Groups *Presolve*

`dparam.presolve_tol_rel_lindep`

Relative tolerance employed by the linear dependency checker.

Default 1.0e-10

Accepted [0.0; +inf]

Example `task.putdouparam(dparam.presolve_tol_rel_lindep, 1.0e-10)`

Generic name MSK_DPAR_PRESOLVE_TOL_REL_LINDEP

Groups *Presolve*

`dparam.presolve_tol_s`

Absolute zero tolerance employed for s_i in the presolve.

Default 1.0e-8

Accepted [0.0; +inf]

Example `task.putdouparam(dparam.presolve_tol_s, 1.0e-8)`

Generic name MSK_DPAR_PRESOLVE_TOL_S

Groups *Presolve*

`dparam.presolve_tol_x`

Absolute zero tolerance employed for x_j in the presolve.

Default 1.0e-8

Accepted [0.0; +inf]

Example `task.putdouparam(dparam.presolve_tol_x, 1.0e-8)`

Generic name MSK_DPAR_PRESOLVE_TOL_X

Groups *Presolve*

`dparam.qcqo_reformulate_rel_drop_tol`

This parameter determines when columns are dropped in incomplete Cholesky factorization during reformulation of quadratic problems.

Default 1e-15

Accepted [0; +inf]

Example `task.putdouparam(dparam.qcqo_reformulate_rel_drop_tol, 1e-15)`

Generic name MSK_DPAR_QCQO_REFORMULATE_REL_DROP_TOL

Groups *Interior-point method*

`dparam.semidefinite_tol_approx`

Tolerance to define a matrix to be positive semidefinite.

Default 1.0e-10

Accepted [1.0e-15; +inf]

Example `task.putdouparam(dparam.semidefinite_tol_approx, 1.0e-10)`

Generic name MSK_DPAR_SEMIDEFINITE_TOL_APPROX

Groups *Data check*

`dparam.sim_lu_tol_rel_piv`

Relative pivot tolerance employed when computing the LU factorization of the basis in the simplex optimizers and in the basis identification procedure. A value closer to 1.0 generally improves numerical stability but typically also implies an increase in the computational work.

Default 0.01

Accepted [1.0e-6; 0.999999]

Example `task.putdouparam(dparam.sim_lu_tol_rel_piv, 0.01)`

Generic name MSK_DPAR_SIM_LU_TOL_REL_PIV

Groups *Basis identification*, *Simplex optimizer*

`dparam.simplex_abs_tol_piv`

Absolute pivot tolerance employed by the simplex optimizers.

Default 1.0e-7

Accepted [1.0e-12; +inf]

Example `task.putdouparam(dparam.simplex_abs_tol_piv, 1.0e-7)`

Generic name MSK_DPAR_SIMPLEX_ABS_TOL_PIV

Groups *Simplex optimizer*

`dparam.upper_obj_cut`

If either a primal or dual feasible solution is found proving that the optimal objective value is outside the interval [*dparam.lower_obj_cut* , *dparam.upper_obj_cut*], then **MOSEK** is terminated.

Default 1.0e30

Accepted [-inf; +inf]

Example `task.putdouparam(dparam.upper_obj_cut, 1.0e30)`

See also *dparam.upper_obj_cut_finite_trh*

Generic name MSK_DPAR_UPPER_OBJ_CUT

Groups *Termination criteria*

`dparam.upper_obj_cut_finite_trh`

If the upper objective cut is greater than the value of this parameter, then the upper objective cut *dparam.upper_obj_cut* is treated as ∞.

Default 0.5e30

Accepted [-inf; +inf]

Example `task.putdouparam(dparam.upper_obj_cut_finite_trh, 0.5e30)`

Generic name MSK_DPAR_UPPER_OBJ_CUT_FINITE_TRH

Groups *Termination criteria*

15.7.2 Integer parameters

`iparam`

The enumeration type containing all integer parameters.

`iparam.ana_sol_basis`

Controls whether the basis matrix is analyzed in solution analyzer.

Default *on*

Accepted *on*, *off* (see *onoffkey*)

Example `task.putintparam(iparam.ana_sol_basis, onoffkey.on)`

Generic name MSK_IPAR_ANA_SOL_BASIS

Groups *Analysis*

`iparam.ana_sol_print_violated`

A parameter of the problem analyzer. Controls whether a list of violated constraints is printed. All constraints violated by more than the value set by the parameter *dparam.ana_sol_infeas_tol* will be printed.

Default *off*

Accepted *on*, *off* (see *onoffkey*)

Example `task.putintparam(iparam.ana_sol_print_violated, onoffkey.off)`

Generic name MSK_IPAR_ANA_SOL_PRINT_VIOLATED

Groups *Analysis*

`iparam.auto_sort_a_before_opt`

Controls whether the elements in each column of A are sorted before an optimization is performed. This is not required but makes the optimization more deterministic.

Default *off*

Accepted *on*, *off* (see *onoffkey*)

Example `task.putintparam(iparam.auto_sort_a_before_opt, onoffkey.off)`

Generic name MSK_IPAR_AUTO_SORT_A_BEFORE_OPT

Groups *Debugging*

`iparam.auto_update_sol_info`

Controls whether the solution information items are automatically updated after an optimization is performed.

Default *off*

Accepted *on*, *off* (see *onoffkey*)

Example `task.putintparam(iparam.auto_update_sol_info, onoffkey.off)`

Generic name MSK_IPAR_AUTO_UPDATE_SOL_INFO

Groups *Overall system*

`iparam.basis_solve_use_plus_one`

If a slack variable is in the basis, then the corresponding column in the basis is a unit vector with -1 in the right position. However, if this parameter is set to *onoffkey.on*, -1 is replaced by 1.

This has significance for the results returned by the *Task.solvewithbasis* function.

Default *off*

Accepted *on*, *off* (see *onoffkey*)

Example `task.putintparam(iparam.basis_solve_use_plus_one, onoffkey.off)`

Generic name MSK_IPAR_BASIS_SOLVE_USE_PLUS_ONE

Groups *Simplex optimizer*

`iparam.bi_clean_optimizer`

Controls which simplex optimizer is used in the clean-up phase. Anything else than *optimizertype.primal_simplex* or *optimizertype.dual_simplex* is equivalent to *optimizertype.free_simplex*.

Default *free*

Accepted *free*, *intpnt*, *conic*, *primal_simplex*, *dual_simplex*, *free_simplex*, *mixed_int* (see *optimizertype*)

Example `task.putintparam(iparam.bi_clean_optimizer, optimizertype.free)`

Generic name MSK_IPAR_BI_CLEAN_OPTIMIZER

Groups *Basis identification*, *Overall solver*

`iparam.bi_ignore_max_iter`

If the parameter *iparam.intpnt_basis* has the value *basindtype.no_error* and the interior-point optimizer has terminated due to maximum number of iterations, then basis identification is performed if this parameter has the value *onoffkey.on*.

Default *off*

Accepted *on*, *off* (see *onoffkey*)

Example `task.putintparam(iparam.bi_ignore_max_iter, onoffkey.off)`

Generic name MSK_IPAR_BI_IGNORE_MAX_ITER

Groups *Interior-point method*, *Basis identification*

`iparam.bi_ignore_num_error`

If the parameter *iparam.intpnt_basis* has the value *basindtype.no_error* and the interior-point optimizer has terminated due to a numerical problem, then basis identification is performed if this parameter has the value *onoffkey.on*.

Default *off*

Accepted *on*, *off* (see *onoffkey*)

Example `task.putintparam(iparam.bi_ignore_num_error, onoffkey.off)`

Generic name MSK_IPAR_BI_IGNORE_NUM_ERROR

Groups *Interior-point method*, *Basis identification*

`iparam.bi_max_iterations`

Controls the maximum number of simplex iterations allowed to optimize a basis after the basis identification.

Default 1000000

Accepted [0; +inf]

Example `task.putintparam(iparam.bi_max_iterations, 1000000)`

Generic name MSK_IPAR_BI_MAX_ITERATIONS

Groups *Basis identification*, *Termination criteria*

`iparam.cache_license`

Specifies if the license is kept checked out for the lifetime of the **MOSEK** environment/model/process (*onoffkey.on*) or returned to the server immediately after the optimization (*onoffkey.off*).

By default the license is checked out for the lifetime of the **MOSEK** environment by the first call to *Task.optimize*.

Check-in and check-out of licenses have an overhead. Frequent communication with the license server should be avoided.

Default *on*

Accepted *on*, *off* (see *onoffkey*)

Example `task.putintparam(iparam.cache_license, onoffkey.on)`

Generic name MSK_IPAR_CACHE_LICENSE

Groups *License manager*

`iparam.check_convexity`

Specify the level of convexity check on quadratic problems.

Default *full*

Accepted *none*, *simple*, *full* (see *checkconvexitytype*)

Example `task.putintparam(iparam.check_convexity, checkconvexitytype.full)`

Generic name MSK_IPAR_CHECK_CONVEXITY

Groups *Data check*

`iparam.compress_statfile`

Control compression of stat files.

Default *on*

Accepted *on*, *off* (see *onoffkey*)

Example `task.putintparam(iparam.compress_statfile, onoffkey.on)`

Generic name MSK_IPAR_COMPRESS_STATFILE

`iparam.infeas_generic_names`

Controls whether generic names are used when an infeasible subproblem is created.

Default *off*

Accepted *on*, *off* (see *onoffkey*)

Example `task.putintparam(iparam.infeas_generic_names, onoffkey.off)`

Generic name MSK_IPAR_INFEAS_GENERIC_NAMES

Groups *Infeasibility report*

`iparam.infeas_prefer_primal`

If both certificates of primal and dual infeasibility are supplied then only the primal is used when this option is turned on.

Default *on*

Accepted *on*, *off* (see *onoffkey*)

Example `task.putintparam(iparam.infeas_prefer_primal, onoffkey.on)`

Generic name MSK_IPAR_INFEAS_PREFER_PRIMAL

Groups *Overall solver*

`iparam.infeas_report_auto`

Controls whether an infeasibility report is automatically produced after the optimization if the problem is primal or dual infeasible.

Default *off*

Accepted *on*, *off* (see *onoffkey*)

Example `task.putintparam(iparam.infeas_report_auto, onoffkey.off)`

Generic name MSK_IPAR_INFEAS_REPORT_AUTO

Groups *Data input/output*, *Solution input/output*

`iparam.infeas_report_level`

Controls the amount of information presented in an infeasibility report. Higher values imply more information.

Default 1

Accepted [0; +inf]

Example `task.putintparam(iparam.infeas_report_level, 1)`

Generic name MSK_IPAR_INFEAS_REPORT_LEVEL

Groups *Infeasibility report*, *Output information*

`iparam.intpnt_basis`

Controls whether the interior-point optimizer also computes an optimal basis.

Default *always*

Accepted *never*, *always*, *no_error*, *if_feasible*, *reservered* (see *basindtype*)

Example `task.putintparam(iparam.intpnt_basis, basindtype.always)`

See also *iparam.bi_ignore_max_iter*, *iparam.bi_ignore_num_error*, *iparam.bi_max_iterations*, *iparam.bi_clean_optimizer*

Generic name MSK_IPAR_INTPNT_BASIS

Groups *Interior-point method*, *Basis identification*

`iparam.intpnt_diff_step`

Controls whether different step sizes are allowed in the primal and dual space.

Default *on*

Accepted

- *on*: Different step sizes are allowed.
- *off*: Different step sizes are not allowed.

Example `task.putintparam(iparam.intpnt_diff_step, onoffkey.on)`

Generic name MSK_IPAR_INTPNT_DIFF_STEP

Groups *Interior-point method*

`iparam.intpnt_hotstart`

Currently not in use.

Default *none*

Accepted *none*, *primal*, *dual*, *primal_dual* (see *intpnthotstart*)

Example `task.putintparam(iparam.intpnt_hotstart, intpnthotstart.none)`

Generic name MSK_IPAR_INTPNT_HOTSTART

Groups *Interior-point method*

`iparam.intpnt_max_iterations`

Controls the maximum number of iterations allowed in the interior-point optimizer.

Default 400

Accepted [0; +inf]

Example `task.putintparam(iparam.intpnt_max_iterations, 400)`

Generic name MSK_IPAR_INTPNT_MAX_ITERATIONS

Groups *Interior-point method*, *Termination criteria*

`iparam.intpnt_max_num_cor`

Controls the maximum number of correctors allowed by the multiple corrector procedure. A negative value means that **MOSEK** is making the choice.

Default -1

Accepted [-1; +inf]

Example `task.putintparam(iparam.intpnt_max_num_cor, -1)`

Generic name MSK_IPAR_INTPNT_MAX_NUM_COR

Groups *Interior-point method*

`iparam.intpnt_max_num_refinement_steps`

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.

Default -1

Accepted [-inf; +inf]

Example `task.putintparam(iparam.intpnt_max_num_refinement_steps, -1)`

Generic name MSK_IPAR_INTPNT_MAX_NUM_REFINEMENT_STEPS
Groups *Interior-point method*

`iparam.intpnt_multi_thread`
Controls whether the interior-point optimizers are allowed to employ multiple threads if more threads is available.

Default *on*
Accepted *on*, *off* (see *onoffkey*)
Example `task.putintparam(iparam.intpnt_multi_thread, onoffkey.on)`
Generic name MSK_IPAR_INTPNT_MULTI_THREAD
Groups *Overall system*

`iparam.intpnt_off_col_trh`
Controls how many offending columns are detected in the Jacobian of the constraint matrix.

0	no detection
1	aggressive detection
>1	higher values mean less aggressive detection

Default 40
Accepted [0; +inf]
Example `task.putintparam(iparam.intpnt_off_col_trh, 40)`
Generic name MSK_IPAR_INTPNT_OFF_COL_TRH
Groups *Interior-point method*

`iparam.intpnt_order_gp_num_seeds`
The GP ordering is dependent on a random seed. Therefore, trying several random seeds may lead to a better ordering. This parameter controls the number of random seeds tried.

A value of 0 means that MOSEK makes the choice.

Default 0
Accepted [0; +inf]
Example `task.putintparam(iparam.intpnt_order_gp_num_seeds, 0)`
Generic name MSK_IPAR_INTPNT_ORDER_GP_NUM_SEEDS
Groups *Interior-point method*

`iparam.intpnt_order_method`
Controls the ordering strategy used by the interior-point optimizer when factorizing the Newton equation system.

Default *free*
Accepted *free*, *appminloc*, *experimental*, *try_graphpar*, *force_graphpar*, *none* (see *orderingtype*)
Example `task.putintparam(iparam.intpnt_order_method, orderingtype.free)`
Generic name MSK_IPAR_INTPNT_ORDER_METHOD
Groups *Interior-point method*

`iparam.intpnt_purify`
Currently not in use.

Default *none*
Accepted *none*, *primal*, *dual*, *primal_dual*, *auto* (see *purify*)
Example `task.putintparam(iparam.intpnt_purify, purify.none)`
Generic name MSK_IPAR_INTPNT_PURIFY
Groups *Interior-point method*

iparam.intpnt_regularization_use
Controls whether regularization is allowed.

Default *on*
Accepted *on*, *off* (see *onoffkey*)
Example `task.putintparam(iparam.intpnt_regularization_use, onoffkey.on)`
Generic name MSK_IPAR_INTPNT_REGULARIZATION_USE
Groups *Interior-point method*

iparam.intpnt_scaling
Controls how the problem is scaled before the interior-point optimizer is used.

Default *free*
Accepted *free*, *none*, *moderate*, *aggressive* (see *scalingtype*)
Example `task.putintparam(iparam.intpnt_scaling, scalingtype.free)`
Generic name MSK_IPAR_INTPNT_SCALING
Groups *Interior-point method*

iparam.intpnt_solve_form
Controls whether the primal or the dual problem is solved.

Default *free*
Accepted *free*, *primal*, *dual* (see *solveform*)
Example `task.putintparam(iparam.intpnt_solve_form, solveform.free)`
Generic name MSK_IPAR_INTPNT_SOLVE_FORM
Groups *Interior-point method*

iparam.intpnt_starting_point
Starting point used by the interior-point optimizer.

Default *free*
Accepted *free*, *guess*, *constant*, *satisfy_bounds* (see *startpointtype*)
Example `task.putintparam(iparam.intpnt_starting_point, startpointtype.free)`
Generic name MSK_IPAR_INTPNT_STARTING_POINT
Groups *Interior-point method*

iparam.license_debug
This option is used to turn on debugging of the license manager.

Default *off*
Accepted *on*, *off* (see *onoffkey*)
Example `task.putintparam(iparam.license_debug, onoffkey.off)`
Generic name MSK_IPAR_LICENSE_DEBUG
Groups *License manager*

iparam.license_pause_time
If *iparam.license_wait* is *onoffkey.on* and no license is available, then **MOSEK** sleeps a number of milliseconds between each check of whether a license has become free.

Default 100
Accepted [0; 1000000]
Example `task.putintparam(iparam.license_pause_time, 100)`
Generic name MSK_IPAR_LICENSE_PAUSE_TIME
Groups *License manager*

iparam.license_suppress_expire_wrns
Controls whether license features expire warnings are suppressed.

Default *off*

Accepted *on*, *off* (see *onoffkey*)

Example `task.putintparam(iparam.license_suppress_expire_wrns, onoffkey.off)`

Generic name MSK_IPAR_LICENSE_SUPPRESS_EXPIRE_WRNS

Groups *License manager*, *Output information*

`iparam.license_trh_expiry_wrn`

If a license feature expires in a numbers of days less than the value of this parameter then a warning will be issued.

Default 7

Accepted [0; +inf]

Example `task.putintparam(iparam.license_trh_expiry_wrn, 7)`

Generic name MSK_IPAR_LICENSE_TRH_EXPIRY_WRN

Groups *License manager*, *Output information*

`iparam.license_wait`

If all licenses are in use **MOSEK** returns with an error code. However, by turning on this parameter **MOSEK** will wait for an available license.

Default *off*

Accepted *on*, *off* (see *onoffkey*)

Example `task.putintparam(iparam.license_wait, onoffkey.off)`

Generic name MSK_IPAR_LICENSE_WAIT

Groups *Overall solver*, *Overall system*, *License manager*

`iparam.log`

Controls the amount of log information. The value 0 implies that all log information is suppressed. A higher level implies that more information is logged.

Please note that if a task is employed to solve a sequence of optimization problems the value of this parameter is reduced by the value of *iparam.log_cut_second_opt* for the second and any subsequent optimizations.

Default 10

Accepted [0; +inf]

Example `task.putintparam(iparam.log, 10)`

See also *iparam.log_cut_second_opt*

Generic name MSK_IPAR_LOG

Groups *Output information*, *Logging*

`iparam.log_ana_pro`

Controls amount of output from the problem analyzer.

Default 1

Accepted [0; +inf]

Example `task.putintparam(iparam.log_ana_pro, 1)`

Generic name MSK_IPAR_LOG_ANA_PRO

Groups *Analysis*, *Logging*

`iparam.log_bi`

Controls the amount of output printed by the basis identification procedure. A higher level implies that more information is logged.

Default 1

Accepted [0; +inf]

Example `task.putintparam(iparam.log_bi, 1)`

Generic name MSK_IPAR_LOG_BI

Groups *Basis identification*, *Output information*, *Logging*

`iparam.log_bi_freq`
Controls how frequently the optimizer outputs information about the basis identification and how frequent the user-defined callback function is called.

Default 2500
Accepted [0; +inf]
Example `task.putintparam(iparam.log_bi_freq, 2500)`
Generic name MSK_IPAR_LOG_BI_FREQ
Groups *Basis identification*, *Output information*, *Logging*

`iparam.log_check_convexity`
Controls logging in convexity check on quadratic problems. Set to a positive value to turn logging on. If a quadratic coefficient matrix is found to violate the requirement of PSD (NSD) then a list of negative (positive) pivot elements is printed. The absolute value of the pivot elements is also shown.

Default 0
Accepted [0; +inf]
Example `task.putintparam(iparam.log_check_convexity, 0)`
Generic name MSK_IPAR_LOG_CHECK_CONVEXITY
Groups *Data check*

`iparam.log_cut_second_opt`
If a task is employed to solve a sequence of optimization problems, then the value of the log levels is reduced by the value of this parameter. E.g *`iparam.log`* and *`iparam.log_sim`* are reduced by the value of this parameter for the second and any subsequent optimizations.

Default 1
Accepted [0; +inf]
Example `task.putintparam(iparam.log_cut_second_opt, 1)`
See also *`iparam.log`*, *`iparam.log_intpnt`*, *`iparam.log_mio`*, *`iparam.log_sim`*
Generic name MSK_IPAR_LOG_CUT_SECOND_OPT
Groups *Output information*, *Logging*

`iparam.log_expand`
Controls the amount of logging when a data item such as the maximum number constrains is expanded.

Default 0
Accepted [0; +inf]
Example `task.putintparam(iparam.log_expand, 0)`
Generic name MSK_IPAR_LOG_EXPAND
Groups *Output information*, *Logging*

`iparam.log_feas_repair`
Controls the amount of output printed when performing feasibility repair. A value higher than one means extensive logging.

Default 1
Accepted [0; +inf]
Example `task.putintparam(iparam.log_feas_repair, 1)`
Generic name MSK_IPAR_LOG_FEAS_REPAIR
Groups *Output information*, *Logging*

`iparam.log_file`
If turned on, then some log info is printed when a file is written or read.

Default 1

Accepted [0; +inf]

Example `task.putintparam(iparam.log_file, 1)`

Generic name MSK_IPAR_LOG_FILE

Groups *Data input/output*, *Output information*, *Logging*

`iparam.log_include_summary`

If on, then the solution summary will be printed by *`Task.optimize`*, so a separate call to *`Task.solutionsummary`* is not necessary.

Default *`off`*

Accepted *`on`*, *`off`* (see *`onoffkey`*)

Example `task.putintparam(iparam.log_include_summary, onoffkey.off)`

Generic name MSK_IPAR_LOG_INCLUDE_SUMMARY

Groups *Output information*, *Logging*

`iparam.log_infeas_ana`

Controls amount of output printed by the infeasibility analyzer procedures. A higher level implies that more information is logged.

Default 1

Accepted [0; +inf]

Example `task.putintparam(iparam.log_infeas_ana, 1)`

Generic name MSK_IPAR_LOG_INFEAS_ANA

Groups *Infeasibility report*, *Output information*, *Logging*

`iparam.log_intpnt`

Controls amount of output printed by the interior-point optimizer. A higher level implies that more information is logged.

Default 1

Accepted [0; +inf]

Example `task.putintparam(iparam.log_intpnt, 1)`

Generic name MSK_IPAR_LOG_INTPNT

Groups *Interior-point method*, *Output information*, *Logging*

`iparam.log_local_info`

Controls whether local identifying information like environment variables, filenames, IP addresses etc. are printed to the log.

Note that this will only affect some functions. Some functions that specifically emit system information will not be affected.

Default *`on`*

Accepted *`on`*, *`off`* (see *`onoffkey`*)

Example `task.putintparam(iparam.log_local_info, onoffkey.on)`

Generic name MSK_IPAR_LOG_LOCAL_INFO

Groups *Output information*, *Logging*

`iparam.log_mio`

Controls the log level for the mixed-integer optimizer. A higher level implies that more information is logged.

Default 4

Accepted [0; +inf]

Example `task.putintparam(iparam.log_mio, 4)`

Generic name MSK_IPAR_LOG_MIO

Groups *Mixed-integer optimization*, *Output information*, *Logging* iparam.log_mio_freq

Controls how frequent the mixed-integer optimizer prints the log line. It will print line every time *iparam.log_mio_freq* relaxations have been solved.

Default 10

Accepted [-inf; +inf]

Example `task.putintparam(iparam.log_mio_freq, 10)`

Generic name MSK_IPAR_LOG_MIO_FREQ

Groups *Mixed-integer optimization*, *Output information*, *Logging*

iparam.log_order

If turned on, then factor lines are added to the log.

Default 1

Accepted [0; +inf]

Example `task.putintparam(iparam.log_order, 1)`

Generic name MSK_IPAR_LOG_ORDER

Groups *Output information*, *Logging*

iparam.log_presolve

Controls amount of output printed by the presolve procedure. A higher level implies that more information is logged.

Default 1

Accepted [0; +inf]

Example `task.putintparam(iparam.log_presolve, 1)`

Generic name MSK_IPAR_LOG_PRESOLVE

Groups *Logging*

iparam.log_response

Controls amount of output printed when response codes are reported. A higher level implies that more information is logged.

Default 0

Accepted [0; +inf]

Example `task.putintparam(iparam.log_response, 0)`

Generic name MSK_IPAR_LOG_RESPONSE

Groups *Output information*, *Logging*

iparam.log_sensitivity

Controls the amount of logging during the sensitivity analysis.

- 0. Means no logging information is produced.
- 1. Timing information is printed.
- 2. Sensitivity results are printed.

Default 1

Accepted [0; +inf]

Example `task.putintparam(iparam.log_sensitivity, 1)`

Generic name MSK_IPAR_LOG_SENSITIVITY

Groups *Output information*, *Logging*

iparam.log_sensitivity_opt

Controls the amount of logging from the optimizers employed during the sensitivity analysis. 0 means no logging information is produced.

Default 0

Accepted [0; +inf]

Example `task.putintparam(iparam.log_sensitivity_opt, 0)`

Generic name MSK_IPAR_LOG_SENSITIVITY_OPT

Groups *Output information*, *Logging*

`iparam.log_sim`

Controls amount of output printed by the simplex optimizer. A higher level implies that more information is logged.

Default 4

Accepted [0; +inf]

Example `task.putintparam(iparam.log_sim, 4)`

Generic name MSK_IPAR_LOG_SIM

Groups *Simplex optimizer*, *Output information*, *Logging*

`iparam.log_sim_freq`

Controls how frequent the simplex optimizer outputs information about the optimization and how frequent the user-defined callback function is called.

Default 1000

Accepted [0; +inf]

Example `task.putintparam(iparam.log_sim_freq, 1000)`

Generic name MSK_IPAR_LOG_SIM_FREQ

Groups *Simplex optimizer*, *Output information*, *Logging*

`iparam.log_sim_minor`

Currently not in use.

Default 1

Accepted [0; +inf]

Example `task.putintparam(iparam.log_sim_minor, 1)`

Generic name MSK_IPAR_LOG_SIM_MINOR

Groups *Simplex optimizer*, *Output information*

`iparam.log_storage`

When turned on, **MOSEK** prints messages regarding the storage usage and allocation.

Default 0

Accepted [0; +inf]

Example `task.putintparam(iparam.log_storage, 0)`

Generic name MSK_IPAR_LOG_STORAGE

Groups *Output information*, *Overall system*, *Logging*

`iparam.max_num_warnings`

Each warning is shown a limited number of times controlled by this parameter. A negative value is identical to infinite number of times.

Default 10

Accepted [-inf; +inf]

Example `task.putintparam(iparam.max_num_warnings, 10)`

Generic name MSK_IPAR_MAX_NUM_WARNINGS

Groups *Output information*

`iparam.mio_branch_dir`

Controls whether the mixed-integer optimizer is branching up or down by default.

Default *free*

Accepted *free*, *up*, *down*, *near*, *far*, *root_lp*, *guided*, *pseudocost* (see *branchdir*)

Example `task.putintparam(iparam.mio_branch_dir, branchdir.free)`

Generic name MSK_IPAR_MIO_BRANCH_DIR
Groups *Mixed-integer optimization*

`iparam.mio_conic_outer_approximation`
If this option is turned on outer approximation is used when solving relaxations of conic problems; otherwise interior point is used.

Default *off*
Accepted *on*, *off* (see *onoffkey*)
Example `task.putintparam(iparam.mio_conic_outer_approximation, onoffkey.off)`
Generic name MSK_IPAR_MIO_CONIC_OUTER_APPROXIMATION
Groups *Mixed-integer optimization*

`iparam.mio_cut_clique`
Controls whether clique cuts should be generated.

Default *on*
Accepted *on*, *off* (see *onoffkey*)
Example `task.putintparam(iparam.mio_cut_clique, onoffkey.on)`
Generic name MSK_IPAR_MIO_CUT_CLIQUE
Groups *Mixed-integer optimization*

`iparam.mio_cut_cmir`
Controls whether mixed integer rounding cuts should be generated.

Default *on*
Accepted *on*, *off* (see *onoffkey*)
Example `task.putintparam(iparam.mio_cut_cmir, onoffkey.on)`
Generic name MSK_IPAR_MIO_CUT_CMIR
Groups *Mixed-integer optimization*

`iparam.mio_cut_gmi`
Controls whether GMI cuts should be generated.

Default *on*
Accepted *on*, *off* (see *onoffkey*)
Example `task.putintparam(iparam.mio_cut_gmi, onoffkey.on)`
Generic name MSK_IPAR_MIO_CUT_GMI
Groups *Mixed-integer optimization*

`iparam.mio_cut_implied_bound`
Controls whether implied bound cuts should be generated.

Default *off*
Accepted *on*, *off* (see *onoffkey*)
Example `task.putintparam(iparam.mio_cut_implied_bound, onoffkey.off)`
Generic name MSK_IPAR_MIO_CUT_IMPLIED_BOUND
Groups *Mixed-integer optimization*

`iparam.mio_cut_knapsack_cover`
Controls whether knapsack cover cuts should be generated.

Default *off*
Accepted *on*, *off* (see *onoffkey*)
Example `task.putintparam(iparam.mio_cut_knapsack_cover, onoffkey.off)`
Generic name MSK_IPAR_MIO_CUT_KNAPSACK_COVER
Groups *Mixed-integer optimization*

iparam.mio_cut_selection_level
Controls how aggressively generated cuts are selected to be included in the relaxation.

- -1. The optimizer chooses the level of cut selection
- 0. Generated cuts less likely to be added to the relaxation
- 1. Cuts are more aggressively selected to be included in the relaxation

Default -1
Accepted [-1; +1]
Example `task.putintparam(iparam.mio_cut_selection_level, -1)`
Generic name MSK_IPAR_MIO_CUT_SELECTION_LEVEL
Groups *Mixed-integer optimization*

iparam.mio_feaspump_level
Controls the way the Feasibility Pump heuristic is employed by the mixed-integer optimizer.

- -1. The optimizer chooses how the Feasibility Pump is used
- 0. The Feasibility Pump is disabled
- 1. The Feasibility Pump is enabled with an effort to improve solution quality
- 2. The Feasibility Pump is enabled with an effort to reach feasibility early

Default -1
Accepted [-1; 2]
Example `task.putintparam(iparam.mio_feaspump_level, -1)`
Generic name MSK_IPAR_MIO_FEASPUMP_LEVEL
Groups *Mixed-integer optimization*

iparam.mio_heuristic_level
Controls the heuristic employed by the mixed-integer optimizer to locate an initial good integer feasible solution. A value of zero means the heuristic is not used at all. A larger value than 0 means that a gradually more sophisticated heuristic is used which is computationally more expensive. A negative value implies that the optimizer chooses the heuristic. Normally a value around 3 to 5 should be optimal.

Default -1
Accepted [-inf; +inf]
Example `task.putintparam(iparam.mio_heuristic_level, -1)`
Generic name MSK_IPAR_MIO_HEURISTIC_LEVEL
Groups *Mixed-integer optimization*

iparam.mio_max_num_branches
Maximum number of branches allowed during the branch and bound search. A negative value means infinite.

Default -1
Accepted [-inf; +inf]
Example `task.putintparam(iparam.mio_max_num_branches, -1)`
Generic name MSK_IPAR_MIO_MAX_NUM_BRANCHES
Groups *Mixed-integer optimization*, *Termination criteria*

iparam.mio_max_num_relaxs
Maximum number of relaxations allowed during the branch and bound search. A negative value means infinite.

Default -1
Accepted [-inf; +inf]
Example `task.putintparam(iparam.mio_max_num_relaxs, -1)`
Generic name MSK_IPAR_MIO_MAX_NUM_RELAXS

Groups *Mixed-integer optimization*

`iparam.mio_max_num_root_cut_rounds`
Maximum number of cut separation rounds at the root node.

Default 100
Accepted [0; +inf]
Example `task.putintparam(iparam.mio_max_num_root_cut_rounds, 100)`
Generic name MSK_IPAR_MIO_MAX_NUM_ROOT_CUT_ROUNDS
Groups *Mixed-integer optimization*, *Termination criteria*

`iparam.mio_max_num_solutions`
The mixed-integer optimizer can be terminated after a certain number of different feasible solutions has been located. If this parameter has the value $n > 0$, then the mixed-integer optimizer will be terminated when n feasible solutions have been located.

Default -1
Accepted [-inf; +inf]
Example `task.putintparam(iparam.mio_max_num_solutions, -1)`
Generic name MSK_IPAR_MIO_MAX_NUM_SOLUTIONS
Groups *Mixed-integer optimization*, *Termination criteria*

`iparam.mio_mode`
Controls whether the optimizer includes the integer restrictions when solving a (mixed) integer optimization problem.

Default *satisfied*
Accepted *ignored*, *satisfied* (see *miomode*)
Example `task.putintparam(iparam.mio_mode, miomode.satisfied)`
Generic name MSK_IPAR_MIO_MODE
Groups *Overall solver*

`iparam.mio_node_optimizer`
Controls which optimizer is employed at the non-root nodes in the mixed-integer optimizer.

Default *free*
Accepted *free*, *intpnt*, *conic*, *primal_simplex*, *dual_simplex*, *free_simplex*, *mixed_int* (see *optimizertype*)
Example `task.putintparam(iparam.mio_node_optimizer, optimizertype.free)`
Generic name MSK_IPAR_MIO_NODE_OPTIMIZER
Groups *Mixed-integer optimization*

`iparam.mio_node_selection`
Controls the node selection strategy employed by the mixed-integer optimizer.

Default *free*
Accepted *free*, *first*, *best*, *pseudo* (see *mionodeseltype*)
Example `task.putintparam(iparam.mio_node_selection, mionodeseltype.free)`
Generic name MSK_IPAR_MIO_NODE_SELECTION
Groups *Mixed-integer optimization*

`iparam.mio_perspective_reformulate`
Enables or disables perspective reformulation in presolve.

Default *on*
Accepted *on*, *off* (see *onoffkey*)
Example `task.putintparam(iparam.mio_perspective_reformulate, onoffkey.on)`

Generic name MSK_IPAR_MIO_PERSPECTIVE_REFORMULATE
Groups *Mixed-integer optimization*

`iparam.mio_probing_level`
Controls the amount of probing employed by the mixed-integer optimizer in presolve.

- −1. The optimizer chooses the level of probing employed
- 0. Probing is disabled
- 1. A low amount of probing is employed
- 2. A medium amount of probing is employed
- 3. A high amount of probing is employed

Default -1
Accepted [-1; 3]
Example `task.putintparam(iparam.mio_probing_level, -1)`
Generic name MSK_IPAR_MIO_PROBING_LEVEL
Groups *Mixed-integer optimization*

`iparam.mio_propagate_objective_constraint`
Use objective domain propagation.

Default *off*
Accepted *on*, *off* (see *onoffkey*)
Example `task.putintparam(iparam.mio_propagate_objective_constraint, onoffkey.off)`
Generic name MSK_IPAR_MIO_PROPAGATE_OBJECTIVE_CONSTRAINT
Groups *Mixed-integer optimization*

`iparam.mio_rins_max_nodes`
Controls the maximum number of nodes allowed in each call to the RINS heuristic. The default value of -1 means that the value is determined automatically. A value of zero turns off the heuristic.

Default -1
Accepted [-1; +inf]
Example `task.putintparam(iparam.mio_rins_max_nodes, -1)`
Generic name MSK_IPAR_MIO_RINS_MAX_NODES
Groups *Mixed-integer optimization*

`iparam.mio_root_optimizer`
Controls which optimizer is employed at the root node in the mixed-integer optimizer.

Default *free*
Accepted *free*, *intpnt*, *conic*, *primal_simplex*, *dual_simplex*, *free_simplex*, *mixed_int* (see *optimizertype*)
Example `task.putintparam(iparam.mio_root_optimizer, optimizertype.free)`
Generic name MSK_IPAR_MIO_ROOT_OPTIMIZER
Groups *Mixed-integer optimization*

`iparam.mio_root_repeat_presolve_level`
Controls whether presolve can be repeated at root node.

- −1. The optimizer chooses whether presolve is repeated
- 0. Never repeat presolve
- 1. Always repeat presolve

Default -1
Accepted [-1; 1]
Example `task.putintparam(iparam.mio_root_repeat_presolve_level, -1)`

Generic name MSK_IPAR_MIO_ROOT_REPEAT_PRESOLVE_LEVEL
Groups *Mixed-integer optimization*

`iparam.mio_seed`
Sets the random seed used for randomization in the mixed integer optimizer. Selecting a different seed can change the path the optimizer takes to the optimal solution.

Default 42
Accepted [0; +inf]
Example `task.putintparam(iparam.mio_seed, 42)`
Generic name MSK_IPAR_MIO_SEED
Groups *Mixed-integer optimization*

`iparam.mio_vb_detection_level`
Controls how much effort is put into detecting variable bounds.

- -1. The optimizer chooses
- 0. No variable bounds are detected
- 1. Only detect variable bounds that are directly represented in the problem
- 2. Detect variable bounds in probing

Default -1
Accepted [-1; +2]
Example `task.putintparam(iparam.mio_vb_detection_level, -1)`
Generic name MSK_IPAR_MIO_VB_DETECTION_LEVEL
Groups *Mixed-integer optimization*

`iparam.mt_spincount`
Set the number of iterations to spin before sleeping.

Default 0
Accepted [0; 1000000000]
Example `task.putintparam(iparam.mt_spincount, 0)`
Generic name MSK_IPAR_MT_SPINCOUNT
Groups *Overall system*

`iparam.num_threads`
Controls the number of threads employed by the optimizer. If set to 0 the number of threads used will be equal to the number of cores detected on the machine.

If using the conic optimizer, the value of this parameter set at first optimization remains constant through the lifetime of the process. **MOSEK** will allocate a thread pool of given size, and changing the parameter value later will have no effect. It will, however, remain possible to demand single-threaded execution by setting *`iparam.intpnt_multi_thread`*.

For the mixed-integer optimizer and interior-point linear optimizer there is no such restriction.

Default 0
Accepted [0; +inf]
Example `task.putintparam(iparam.num_threads, 0)`
Generic name MSK_IPAR_NUM_THREADS
Groups *Overall system*

`iparam.opf_write_header`
Write a text header with date and **MOSEK** version in an OPF file.

Default *`on`*
Accepted *`on`*, *`off`* (see *`onoffkey`*)
Example `task.putintparam(iparam.opf_write_header, onoffkey.on)`

Generic name MSK_IPAR_OPF_WRITE_HEADER
Groups *Data input/output*

iparam.opf_write_hints
Write a hint section with problem dimensions in the beginning of an OPF file.

Default *on*
Accepted *on*, *off* (see *onoffkey*)
Example `task.putintparam(iparam.opf_write_hints, onoffkey.on)`
Generic name MSK_IPAR_OPF_WRITE_HINTS
Groups *Data input/output*

iparam.opf_write_line_length
Aim to keep lines in OPF files not much longer than this.

Default 80
Accepted [0; +inf]
Example `task.putintparam(iparam.opf_write_line_length, 80)`
Generic name MSK_IPAR_OPF_WRITE_LINE_LENGTH
Groups *Data input/output*

iparam.opf_write_parameters
Write a parameter section in an OPF file.

Default *off*
Accepted *on*, *off* (see *onoffkey*)
Example `task.putintparam(iparam.opf_write_parameters, onoffkey.off)`
Generic name MSK_IPAR_OPF_WRITE_PARAMETERS
Groups *Data input/output*

iparam.opf_write_problem
Write objective, constraints, bounds etc. to an OPF file.

Default *on*
Accepted *on*, *off* (see *onoffkey*)
Example `task.putintparam(iparam.opf_write_problem, onoffkey.on)`
Generic name MSK_IPAR_OPF_WRITE_PROBLEM
Groups *Data input/output*

iparam.opf_write_sol_bas
If *iparam.opf_write_solutions* is *onoffkey.on* and a basic solution is defined, include the basic solution in OPF files.

Default *on*
Accepted *on*, *off* (see *onoffkey*)
Example `task.putintparam(iparam.opf_write_sol_bas, onoffkey.on)`
Generic name MSK_IPAR_OPF_WRITE_SOL_BAS
Groups *Data input/output*

iparam.opf_write_sol_itg
If *iparam.opf_write_solutions* is *onoffkey.on* and an integer solution is defined, write the integer solution in OPF files.

Default *on*
Accepted *on*, *off* (see *onoffkey*)
Example `task.putintparam(iparam.opf_write_sol_itg, onoffkey.on)`
Generic name MSK_IPAR_OPF_WRITE_SOL_ITG
Groups *Data input/output*

iparam.opf_write_sol_itr

If *iparam.opf_write_solutions* is *onoffkey.on* and an interior solution is defined, write the interior solution in OPF files.

Default *on*

Accepted *on*, *off* (see *onoffkey*)

Example `task.putintparam(iparam.opf_write_sol_itr, onoffkey.on)`

Generic name MSK_IPAR_OPF_WRITE_SOL_ITR

Groups *Data input/output*

iparam.opf_write_solutions

Enable inclusion of solutions in the OPF files.

Default *off*

Accepted *on*, *off* (see *onoffkey*)

Example `task.putintparam(iparam.opf_write_solutions, onoffkey.off)`

Generic name MSK_IPAR_OPF_WRITE_SOLUTIONS

Groups *Data input/output*

iparam.optimizer

The parameter controls which optimizer is used to optimize the task.

Default *free*

Accepted *free*, *intpnt*, *conic*, *primal_simplex*, *dual_simplex*, *free_simplex*, *mixed_int* (see *optimizertype*)

Example `task.putintparam(iparam.optimizer, optimizertype.free)`

Generic name MSK_IPAR_OPTIMIZER

Groups *Overall solver*

iparam.param_read_case_name

If turned on, then names in the parameter file are case sensitive.

Default *on*

Accepted *on*, *off* (see *onoffkey*)

Example `task.putintparam(iparam.param_read_case_name, onoffkey.on)`

Generic name MSK_IPAR_PARAM_READ_CASE_NAME

Groups *Data input/output*

iparam.param_read_ign_error

If turned on, then errors in parameter settings is ignored.

Default *off*

Accepted *on*, *off* (see *onoffkey*)

Example `task.putintparam(iparam.param_read_ign_error, onoffkey.off)`

Generic name MSK_IPAR_PARAM_READ_IGN_ERROR

Groups *Data input/output*

iparam.presolve_eliminator_max_fill

Controls the maximum amount of fill-in that can be created by one pivot in the elimination phase of the presolve. A negative value means the parameter value is selected automatically.

Default -1

Accepted [-inf; +inf]

Example `task.putintparam(iparam.presolve_eliminator_max_fill, -1)`

Generic name MSK_IPAR_PRESOLVE_ELIMINATOR_MAX_FILL

Groups *Presolve*

`iparam.presolve_eliminator_max_num_tries`

Control the maximum number of times the eliminator is tried. A negative value implies **MOSEK** decides.

Default -1

Accepted [-inf; +inf]

Example `task.putintparam(iparam.presolve_eliminator_max_num_tries, -1)`

Generic name MSK_IPAR_PRESOLVE_ELIMINATOR_MAX_NUM_TRIES

Groups *Presolve*

`iparam.presolve_level`

Currently not used.

Default -1

Accepted [-inf; +inf]

Example `task.putintparam(iparam.presolve_level, -1)`

Generic name MSK_IPAR_PRESOLVE_LEVEL

Groups *Overall solver*, *Presolve*

`iparam.presolve_lindep_abs_work_trh`

Controls linear dependency check in presolve. The linear dependency check is potentially computationally expensive.

Default 100

Accepted [-inf; +inf]

Example `task.putintparam(iparam.presolve_lindep_abs_work_trh, 100)`

Generic name MSK_IPAR_PRESOLVE_LINDEP_ABS_WORK_TRH

Groups *Presolve*

`iparam.presolve_lindep_rel_work_trh`

Controls linear dependency check in presolve. The linear dependency check is potentially computationally expensive.

Default 100

Accepted [-inf; +inf]

Example `task.putintparam(iparam.presolve_lindep_rel_work_trh, 100)`

Generic name MSK_IPAR_PRESOLVE_LINDEP_REL_WORK_TRH

Groups *Presolve*

`iparam.presolve_lindep_use`

Controls whether the linear constraints are checked for linear dependencies.

Default *on*

Accepted *on*, *off* (see *onoffkey*)

Example `task.putintparam(iparam.presolve_lindep_use, onoffkey.on)`

Generic name MSK_IPAR_PRESOLVE_LINDEP_USE

Groups *Presolve*

`iparam.presolve_max_num_pass`

Control the maximum number of times presolve passes over the problem. A negative value implies **MOSEK** decides.

Default -1

Accepted [-inf; +inf]

Example `task.putintparam(iparam.presolve_max_num_pass, -1)`

Generic name MSK_IPAR_PRESOLVE_MAX_NUM_PASS

Groups *Presolve*

iparam.presolve_max_num_reductions

Controls the maximum number of reductions performed by the presolve. The value of the parameter is normally only changed in connection with debugging. A negative value implies that an infinite number of reductions are allowed.

Default -1

Accepted [-inf; +inf]

Example `task.putintparam(iparam.presolve_max_num_reductions, -1)`

Generic name MSK_IPAR_PRESOLVE_MAX_NUM_REDUCTIONS

Groups *Overall solver*, *Presolve*

iparam.presolve_use

Controls whether the presolve is applied to a problem before it is optimized.

Default *free*

Accepted *off*, *on*, *free* (see *presolvemode*)

Example `task.putintparam(iparam.presolve_use, presolvemode.free)`

Generic name MSK_IPAR_PRESOLVE_USE

Groups *Overall solver*, *Presolve*

iparam.primal_repair_optimizer

Controls which optimizer that is used to find the optimal repair.

Default *free*

Accepted *free*, *intpnt*, *conic*, *primal_simplex*, *dual_simplex*, *free_simplex*, *mixed_int* (see *optimizertype*)

Example `task.putintparam(iparam.primal_repair_optimizer, optimizertype.free)`

Generic name MSK_IPAR_PRIMAL_REPAIR_OPTIMIZER

Groups *Overall solver*

iparam.ptf_write_transform

If *iparam.ptf_write_transform* is *onoffkey.on*, constraint blocks with identifiable conic slacks are transformed into conic constraints and the slacks are eliminated.

Default *on*

Accepted *on*, *off* (see *onoffkey*)

Example `task.putintparam(iparam.ptf_write_transform, onoffkey.on)`

Generic name MSK_IPAR_PTF_WRITE_TRANSFORM

Groups *Data input/output*

iparam.read_debug

Turns on additional debugging information when reading files.

Default *off*

Accepted *on*, *off* (see *onoffkey*)

Example `task.putintparam(iparam.read_debug, onoffkey.off)`

Generic name MSK_IPAR_READ_DEBUG

Groups *Data input/output*

iparam.read_keep_free_con

Controls whether the free constraints are included in the problem.

Default *off*

Accepted

- *on*: The free constraints are kept.
- *off*: The free constraints are discarded.

Example `task.putintparam(iparam.read_keep_free_con, onoffkey.off)`

Generic name MSK_IPAR_READ_KEEP_FREE_CON
Groups *Data input/output*

`iparam.read_lp_drop_new_vars_in_bou`
If this option is turned on, **MOSEK** will drop variables that are defined for the first time in the bounds section.

Default *off*
Accepted *on*, *off* (see *onoffkey*)
Example `task.putintparam(iparam.read_lp_drop_new_vars_in_bou, onoffkey.off)`
Generic name MSK_IPAR_READ_LP_DROP_NEW_VARS_IN_BOU
Groups *Data input/output*

`iparam.read_lp_quoted_names`
If a name is in quotes when reading an LP file, the quotes will be removed.

Default *on*
Accepted *on*, *off* (see *onoffkey*)
Example `task.putintparam(iparam.read_lp_quoted_names, onoffkey.on)`
Generic name MSK_IPAR_READ_LP_QUOTED_NAMES
Groups *Data input/output*

`iparam.read_mps_format`
Controls how strictly the MPS file reader interprets the MPS format.

Default *free*
Accepted *strict*, *relaxed*, *free*, *cplex* (see *mpsformat*)
Example `task.putintparam(iparam.read_mps_format, mpsformat.free)`
Generic name MSK_IPAR_READ_MPS_FORMAT
Groups *Data input/output*

`iparam.read_mps_width`
Controls the maximal number of characters allowed in one line of the MPS file.

Default 1024
Accepted [80; +inf]
Example `task.putintparam(iparam.read_mps_width, 1024)`
Generic name MSK_IPAR_READ_MPS_WIDTH
Groups *Data input/output*

`iparam.read_task_ignore_param`
Controls whether **MOSEK** should ignore the parameter setting defined in the task file and use the default parameter setting instead.

Default *off*
Accepted *on*, *off* (see *onoffkey*)
Example `task.putintparam(iparam.read_task_ignore_param, onoffkey.off)`
Generic name MSK_IPAR_READ_TASK_IGNORE_PARAM
Groups *Data input/output*

`iparam.remove_unused_solutions`
Removes unused solutions before the optimization is performed.

Default *off*
Accepted *on*, *off* (see *onoffkey*)
Example `task.putintparam(iparam.remove_unused_solutions, onoffkey.off)`
Generic name MSK_IPAR_REMOVE_UNUSED_SOLUTIONS
Groups *Overall system*

iparam.sensitivity_all
If set to *onoffkey.on*, then *Task.sensitivityreport* analyzes all bounds and variables instead of reading a specification from the file.

Default *off*
Accepted *on*, *off* (see *onoffkey*)
Example `task.putintparam(iparam.sensitivity_all, onoffkey.off)`
Generic name MSK_IPAR_SENSITIVITY_ALL
Groups *Overall solver*

iparam.sensitivity_optimizer
Controls which optimizer is used for optimal partition sensitivity analysis.

Default *free_simplex*
Accepted *free*, *intpnt*, *conic*, *primal_simplex*, *dual_simplex*, *free_simplex*, *mixed_int* (see *optimizertype*)
Example `task.putintparam(iparam.sensitivity_optimizer, optimizertype.free_simplex)`
Generic name MSK_IPAR_SENSITIVITY_OPTIMIZER
Groups *Overall solver*, *Simplex optimizer*

iparam.sensitivity_type
Controls which type of sensitivity analysis is to be performed.

Default *basis*
Accepted *basis* (see *sensitivitytype*)
Example `task.putintparam(iparam.sensitivity_type, sensitivitytype.basis)`
Generic name MSK_IPAR_SENSITIVITY_TYPE
Groups *Overall solver*

iparam.sim_basis_factor_use
Controls whether an LU factorization of the basis is used in a hot-start. Forcing a refactorization sometimes improves the stability of the simplex optimizers, but in most cases there is a performance penalty.

Default *on*
Accepted *on*, *off* (see *onoffkey*)
Example `task.putintparam(iparam.sim_basis_factor_use, onoffkey.on)`
Generic name MSK_IPAR_SIM_BASIS_FACTOR_USE
Groups *Simplex optimizer*

iparam.sim_degen
Controls how aggressively degeneration is handled.

Default *free*
Accepted *none*, *free*, *aggressive*, *moderate*, *minimum* (see *simdegen*)
Example `task.putintparam(iparam.sim_degen, simdegen.free)`
Generic name MSK_IPAR_SIM_DEGEN
Groups *Simplex optimizer*

iparam.sim_dual_crash
Controls whether crashing is performed in the dual simplex optimizer. If this parameter is set to x, then a crash will be performed if a basis consists of more than $(100 - x) \mod f_v$ entries, where f_v is the number of fixed variables.

Default 90
Accepted [0; +inf]
Example `task.putintparam(iparam.sim_dual_crash, 90)`

Generic name MSK_IPAR_SIM_DUAL_CRASH
Groups *Dual simplex*

`iparam.sim_dual_phaseone_method`
An experimental feature.

Default 0
Accepted [0; 10]
Example `task.putintparam(iparam.sim_dual_phaseone_method, 0)`
Generic name MSK_IPAR_SIM_DUAL_PHASEONE_METHOD
Groups *Simplex optimizer*

`iparam.sim_dual_restrict_selection`
The dual simplex optimizer can use a so-called restricted selection/pricing strategy to choose the outgoing variable. Hence, if restricted selection is applied, then the dual simplex optimizer first choose a subset of all the potential outgoing variables. Next, for some time it will choose the outgoing variable only among the subset. From time to time the subset is redefined. A larger value of this parameter implies that the optimizer will be more aggressive in its restriction strategy, i.e. a value of 0 implies that the restriction strategy is not applied at all.

Default 50
Accepted [0; 100]
Example `task.putintparam(iparam.sim_dual_restrict_selection, 50)`
Generic name MSK_IPAR_SIM_DUAL_RESTRICT_SELECTION
Groups *Dual simplex*

`iparam.sim_dual_selection`
Controls the choice of the incoming variable, known as the selection strategy, in the dual simplex optimizer.

Default *free*
Accepted *free*, *full*, *ase*, *devex*, *se*, *partial* (see *simseltype*)
Example `task.putintparam(iparam.sim_dual_selection, simseltype.free)`
Generic name MSK_IPAR_SIM_DUAL_SELECTION
Groups *Dual simplex*

`iparam.sim_exploit_dupvec`
Controls if the simplex optimizers are allowed to exploit duplicated columns.

Default *off*
Accepted *on*, *off*, *free* (see *simdupvec*)
Example `task.putintparam(iparam.sim_exploit_dupvec, simdupvec.off)`
Generic name MSK_IPAR_SIM_EXPLOIT_DUPVEC
Groups *Simplex optimizer*

`iparam.sim_hotstart`
Controls the type of hot-start that the simplex optimizer perform.

Default *free*
Accepted *none*, *free*, *status_keys* (see *simhotstart*)
Example `task.putintparam(iparam.sim_hotstart, simhotstart.free)`
Generic name MSK_IPAR_SIM_HOTSTART
Groups *Simplex optimizer*

`iparam.sim_hotstart_lu`
Determines if the simplex optimizer should exploit the initial factorization.

Default *on*
Accepted

- *on*: Factorization is reused if possible.
- *off*: Factorization is recomputed.

Example `task.putintparam(iparam.sim_hotstart_lu, onoffkey.on)`
Generic name MSK_IPAR_SIM_HOTSTART_LU
Groups *Simplex optimizer*

`iparam.sim_max_iterations`
Maximum number of iterations that can be used by a simplex optimizer.

Default 10000000
Accepted [0; +inf]
Example `task.putintparam(iparam.sim_max_iterations, 10000000)`
Generic name MSK_IPAR_SIM_MAX_ITERATIONS
Groups *Simplex optimizer*, *Termination criteria*

`iparam.sim_max_num_setbacks`
Controls how many set-backs are allowed within a simplex optimizer. A set-back is an event where the optimizer moves in the wrong direction. This is impossible in theory but may happen due to numerical problems.

Default 250
Accepted [0; +inf]
Example `task.putintparam(iparam.sim_max_num_setbacks, 250)`
Generic name MSK_IPAR_SIM_MAX_NUM_SETBACKS
Groups *Simplex optimizer*

`iparam.sim_non_singular`
Controls if the simplex optimizer ensures a non-singular basis, if possible.

Default *on*
Accepted *on*, *off* (see *onoffkey*)
Example `task.putintparam(iparam.sim_non_singular, onoffkey.on)`
Generic name MSK_IPAR_SIM_NON_SINGULAR
Groups *Simplex optimizer*

`iparam.sim_primal_crash`
Controls whether crashing is performed in the primal simplex optimizer. In general, if a basis consists of more than (100-this parameter value)% fixed variables, then a crash will be performed.

Default 90
Accepted [0; +inf]
Example `task.putintparam(iparam.sim_primal_crash, 90)`
Generic name MSK_IPAR_SIM_PRIMAL_CRASH
Groups *Primal simplex*

`iparam.sim_primal_phaseone_method`
An experimental feature.

Default 0
Accepted [0; 10]
Example `task.putintparam(iparam.sim_primal_phaseone_method, 0)`
Generic name MSK_IPAR_SIM_PRIMAL_PHASEONE_METHOD
Groups *Simplex optimizer*

`iparam.sim_primal_restrict_selection`
The primal simplex optimizer can use a so-called restricted selection/pricing strategy to choose the outgoing variable. Hence, if restricted selection is applied, then the primal simplex optimizer first choose a subset of all the potential incoming variables. Next, for some time it will choose the incoming variable only among the subset. From time to time the subset is redefined. A larger value of this parameter implies that the optimizer will be more aggressive in its restriction strategy, i.e. a value of 0 implies that the restriction strategy is not applied at all.

Default 50
Accepted [0; 100]
Example `task.putintparam(iparam.sim_primal_restrict_selection, 50)`
Generic name MSK_IPAR_SIM_PRIMAL_RESTRICT_SELECTION
Groups *Primal simplex*

`iparam.sim_primal_selection`
Controls the choice of the incoming variable, known as the selection strategy, in the primal simplex optimizer.

Default *free*
Accepted *free*, *full*, *ase*, *devex*, *se*, *partial* (see *simseltype*)
Example `task.putintparam(iparam.sim_primal_selection, simseltype.free)`
Generic name MSK_IPAR_SIM_PRIMAL_SELECTION
Groups *Primal simplex*

`iparam.sim_refactor_freq`
Controls how frequent the basis is refactorized. The value 0 means that the optimizer determines the best point of refactorization. It is strongly recommended NOT to change this parameter.

Default 0
Accepted [0; +inf]
Example `task.putintparam(iparam.sim_refactor_freq, 0)`
Generic name MSK_IPAR_SIM_REFACTOR_FREQ
Groups *Simplex optimizer*

`iparam.sim_reformulation`
Controls if the simplex optimizers are allowed to reformulate the problem.

Default *off*
Accepted *on*, *off*, *free*, *aggressive* (see *simreform*)
Example `task.putintparam(iparam.sim_reformulation, simreform.off)`
Generic name MSK_IPAR_SIM_REFORMULATION
Groups *Simplex optimizer*

`iparam.sim_save_lu`
Controls if the LU factorization stored should be replaced with the LU factorization corresponding to the initial basis.

Default *off*
Accepted *on*, *off* (see *onoffkey*)
Example `task.putintparam(iparam.sim_save_lu, onoffkey.off)`
Generic name MSK_IPAR_SIM_SAVE_LU
Groups *Simplex optimizer*

`iparam.sim_scaling`
Controls how much effort is used in scaling the problem before a simplex optimizer is used.

Default *free*
Accepted *free*, *none*, *moderate*, *aggressive* (see *scalingtype*)
Example `task.putintparam(iparam.sim_scaling, scalingtype.free)`
Generic name MSK_IPAR_SIM_SCALING
Groups *Simplex optimizer*

`iparam.sim_scaling_method`
Controls how the problem is scaled before a simplex optimizer is used.

Default *pow2*
Accepted *pow2*, *free* (see *scalingmethod*)

Example `task.putintparam(iparam.sim_scaling_method, scalingmethod.pow2)`

Generic name MSK_IPAR_SIM_SCALING_METHOD

Groups *Simplex optimizer*

`iparam.sim_seed`

Sets the random seed used for randomization in the simplex optimizers.

Default 23456

Accepted [0; 32749]

Example `task.putintparam(iparam.sim_seed, 23456)`

Generic name MSK_IPAR_SIM_SEED

Groups *Simplex optimizer*

`iparam.sim_solve_form`

Controls whether the primal or the dual problem is solved by the primal-/dual-simplex optimizer.

Default *free*

Accepted *free*, *primal*, *dual* (see *solveform*)

Example `task.putintparam(iparam.sim_solve_form, solveform.free)`

Generic name MSK_IPAR_SIM_SOLVE_FORM

Groups *Simplex optimizer*

`iparam.sim_stability_priority`

Controls how high priority the numerical stability should be given.

Default 50

Accepted [0; 100]

Example `task.putintparam(iparam.sim_stability_priority, 50)`

Generic name MSK_IPAR_SIM_STABILITY_PRIORITY

Groups *Simplex optimizer*

`iparam.sim_switch_optimizer`

The simplex optimizer sometimes chooses to solve the dual problem instead of the primal problem. This implies that if you have chosen to use the dual simplex optimizer and the problem is dualized, then it actually makes sense to use the primal simplex optimizer instead. If this parameter is on and the problem is dualized and furthermore the simplex optimizer is chosen to be the primal (dual) one, then it is switched to the dual (primal).

Default *off*

Accepted *on*, *off* (see *onoffkey*)

Example `task.putintparam(iparam.sim_switch_optimizer, onoffkey.off)`

Generic name MSK_IPAR_SIM_SWITCH_OPTIMIZER

Groups *Simplex optimizer*

`iparam.sol_filter_keep_basic`

If turned on, then basic and super basic constraints and variables are written to the solution file independent of the filter setting.

Default *off*

Accepted *on*, *off* (see *onoffkey*)

Example `task.putintparam(iparam.sol_filter_keep_basic, onoffkey.off)`

Generic name MSK_IPAR_SOL_FILTER_KEEP_BASIC

Groups *Solution input/output*

`iparam.sol_filter_keep_ranged`

If turned on, then ranged constraints and variables are written to the solution file independent of the filter setting.

Default *off*

Accepted *on*, *off* (see *onoffkey*)
Example `task.putintparam(iparam.sol_filter_keep_ranged, onoffkey.off)`
Generic name MSK_IPAR_SOL_FILTER_KEEP_RANGED
Groups *Solution input/output*

`iparam.sol_read_name_width`
When a solution is read by **MOSEK** and some constraint, variable or cone names contain blanks, then a maximum name width much be specified. A negative value implies that no name contain blanks.

Default -1
Accepted [-inf; +inf]
Example `task.putintparam(iparam.sol_read_name_width, -1)`
Generic name MSK_IPAR_SOL_READ_NAME_WIDTH
Groups *Data input/output*, *Solution input/output*

`iparam.sol_read_width`
Controls the maximal acceptable width of line in the solutions when read by **MOSEK**.

Default 1024
Accepted [80; +inf]
Example `task.putintparam(iparam.sol_read_width, 1024)`
Generic name MSK_IPAR_SOL_READ_WIDTH
Groups *Data input/output*, *Solution input/output*

`iparam.solution_callback`
Indicates whether solution callbacks will be performed during the optimization.

Default *off*
Accepted *on*, *off* (see *onoffkey*)
Example `task.putintparam(iparam.solution_callback, onoffkey.off)`
Generic name MSK_IPAR_SOLUTION_CALLBACK
Groups *Progress callback*, *Overall solver*

`iparam.timing_level`
Controls the amount of timing performed inside **MOSEK**.

Default 1
Accepted [0; +inf]
Example `task.putintparam(iparam.timing_level, 1)`
Generic name MSK_IPAR_TIMING_LEVEL
Groups *Overall system*

`iparam.write_bas_constraints`
Controls whether the constraint section is written to the basic solution file.

Default *on*
Accepted *on*, *off* (see *onoffkey*)
Example `task.putintparam(iparam.write_bas_constraints, onoffkey.on)`
Generic name MSK_IPAR_WRITE_BAS_CONSTRAINTS
Groups *Data input/output*, *Solution input/output*

`iparam.write_bas_head`
Controls whether the header section is written to the basic solution file.

Default *on*
Accepted *on*, *off* (see *onoffkey*)
Example `task.putintparam(iparam.write_bas_head, onoffkey.on)`

Generic name MSK_IPAR_WRITE_BAS_HEAD
Groups *Data input/output*, *Solution input/output*

`iparam.write_bas_variables`
Controls whether the variables section is written to the basic solution file.

Default *on*
Accepted *on*, *off* (see *onoffkey*)
Example `task.putintparam(iparam.write_bas_variables, onoffkey.on)`
Generic name MSK_IPAR_WRITE_BAS_VARIABLES
Groups *Data input/output*, *Solution input/output*

`iparam.write_compression`
Controls whether the data file is compressed while it is written. 0 means no compression while higher values mean more compression.

Default 9
Accepted [0; +inf]
Example `task.putintparam(iparam.write_compression, 9)`
Generic name MSK_IPAR_WRITE_COMPRESSION
Groups *Data input/output*

`iparam.write_data_param`
If this option is turned on the parameter settings are written to the data file as parameters.

Default *off*
Accepted *on*, *off* (see *onoffkey*)
Example `task.putintparam(iparam.write_data_param, onoffkey.off)`
Generic name MSK_IPAR_WRITE_DATA_PARAM
Groups *Data input/output*

`iparam.write_free_con`
Controls whether the free constraints are written to the data file.

Default *on*
Accepted *on*, *off* (see *onoffkey*)
Example `task.putintparam(iparam.write_free_con, onoffkey.on)`
Generic name MSK_IPAR_WRITE_FREE_CON
Groups *Data input/output*

`iparam.write_generic_names`
Controls whether generic names should be used instead of user-defined names when writing to the data file.

Default *off*
Accepted *on*, *off* (see *onoffkey*)
Example `task.putintparam(iparam.write_generic_names, onoffkey.off)`
Generic name MSK_IPAR_WRITE_GENERIC_NAMES
Groups *Data input/output*

`iparam.write_generic_names_io`
Index origin used in generic names.

Default 1
Accepted [0; +inf]
Example `task.putintparam(iparam.write_generic_names_io, 1)`
Generic name MSK_IPAR_WRITE_GENERIC_NAMES_IO
Groups *Data input/output*

iparam.write_ignore_incompatible_items
Controls if the writer ignores incompatible problem items when writing files.

Default *off*

Accepted

- *on*: Ignore items that cannot be written to the current output file format.
- *off*: Produce an error if the problem contains items that cannot the written to the current output file format.

Example `task.putintparam(iparam.write_ignore_incompatible_items, onoffkey.off)`

Generic name MSK_IPAR_WRITE_IGNORE_INCOMPATIBLE_ITEMS

Groups *Data input/output*

iparam.write_int_constraints
Controls whether the constraint section is written to the integer solution file.

Default *on*

Accepted *on*, *off* (see *onoffkey*)

Example `task.putintparam(iparam.write_int_constraints, onoffkey.on)`

Generic name MSK_IPAR_WRITE_INT_CONSTRAINTS

Groups *Data input/output*, *Solution input/output*

iparam.write_int_head
Controls whether the header section is written to the integer solution file.

Default *on*

Accepted *on*, *off* (see *onoffkey*)

Example `task.putintparam(iparam.write_int_head, onoffkey.on)`

Generic name MSK_IPAR_WRITE_INT_HEAD

Groups *Data input/output*, *Solution input/output*

iparam.write_int_variables
Controls whether the variables section is written to the integer solution file.

Default *on*

Accepted *on*, *off* (see *onoffkey*)

Example `task.putintparam(iparam.write_int_variables, onoffkey.on)`

Generic name MSK_IPAR_WRITE_INT_VARIABLES

Groups *Data input/output*, *Solution input/output*

iparam.write_lp_full_obj
Write all variables, including the ones with 0-coefficients, in the objective.

Default *on*

Accepted *on*, *off* (see *onoffkey*)

Example `task.putintparam(iparam.write_lp_full_obj, onoffkey.on)`

Generic name MSK_IPAR_WRITE_LP_FULL_OBJ

Groups *Data input/output*

iparam.write_lp_line_width
Maximum width of line in an LP file written by **MOSEK**.

Default 80

Accepted [40; +inf]

Example `task.putintparam(iparam.write_lp_line_width, 80)`

Generic name MSK_IPAR_WRITE_LP_LINE_WIDTH

Groups *Data input/output*

iparam.write_lp_quoted_names

If this option is turned on, then **MOSEK** will quote invalid LP names when writing an LP file.

Default *on*

Accepted *on*, *off* (see *onoffkey*)

Example `task.putintparam(iparam.write_lp_quoted_names, onoffkey.on)`

Generic name MSK_IPAR_WRITE_LP_QUOTED_NAMES

Groups *Data input/output*

iparam.write_lp_strict_format

Controls whether LP output files satisfy the LP format strictly.

Default *off*

Accepted *on*, *off* (see *onoffkey*)

Example `task.putintparam(iparam.write_lp_strict_format, onoffkey.off)`

Generic name MSK_IPAR_WRITE_LP_STRICT_FORMAT

Groups *Data input/output*

iparam.write_lp_terms_per_line

Maximum number of terms on a single line in an LP file written by **MOSEK**. 0 means unlimited.

Default 10

Accepted [0; +inf]

Example `task.putintparam(iparam.write_lp_terms_per_line, 10)`

Generic name MSK_IPAR_WRITE_LP_TERMS_PER_LINE

Groups *Data input/output*

iparam.write_mps_format

Controls in which format the MPS is written.

Default *free*

Accepted *strict*, *relaxed*, *free*, *cplex* (see *mpsformat*)

Example `task.putintparam(iparam.write_mps_format, mpsformat.free)`

Generic name MSK_IPAR_WRITE_MPS_FORMAT

Groups *Data input/output*

iparam.write_mps_int

Controls if marker records are written to the MPS file to indicate whether variables are integer restricted.

Default *on*

Accepted *on*, *off* (see *onoffkey*)

Example `task.putintparam(iparam.write_mps_int, onoffkey.on)`

Generic name MSK_IPAR_WRITE_MPS_INT

Groups *Data input/output*

iparam.write_precision

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.

Default 15

Accepted [0; +inf]

Example `task.putintparam(iparam.write_precision, 15)`

Generic name MSK_IPAR_WRITE_PRECISION

Groups *Data input/output*

iparam.write_sol_barvariables

Controls whether the symmetric matrix variables section is written to the solution file.

Default *on*
Accepted *on*, *off* (see *onoffkey*)
Example `task.putintparam(iparam.write_sol_barvariables, onoffkey.on)`
Generic name MSK_IPAR_WRITE_SOL_BARVARIABLES
Groups *Data input/output*, *Solution input/output*

`iparam.write_sol_constraints`
Controls whether the constraint section is written to the solution file.

Default *on*
Accepted *on*, *off* (see *onoffkey*)
Example `task.putintparam(iparam.write_sol_constraints, onoffkey.on)`
Generic name MSK_IPAR_WRITE_SOL_CONSTRAINTS
Groups *Data input/output*, *Solution input/output*

`iparam.write_sol_head`
Controls whether the header section is written to the solution file.

Default *on*
Accepted *on*, *off* (see *onoffkey*)
Example `task.putintparam(iparam.write_sol_head, onoffkey.on)`
Generic name MSK_IPAR_WRITE_SOL_HEAD
Groups *Data input/output*, *Solution input/output*

`iparam.write_sol_ignore_invalid_names`
Even if the names are invalid MPS names, then they are employed when writing the solution file.

Default *off*
Accepted *on*, *off* (see *onoffkey*)
Example `task.putintparam(iparam.write_sol_ignore_invalid_names, onoffkey.off)`
Generic name MSK_IPAR_WRITE_SOL_IGNORE_INVALID_NAMES
Groups *Data input/output*, *Solution input/output*

`iparam.write_sol_variables`
Controls whether the variables section is written to the solution file.

Default *on*
Accepted *on*, *off* (see *onoffkey*)
Example `task.putintparam(iparam.write_sol_variables, onoffkey.on)`
Generic name MSK_IPAR_WRITE_SOL_VARIABLES
Groups *Data input/output*, *Solution input/output*

`iparam.write_task_inc_sol`
Controls whether the solutions are stored in the task file too.

Default *on*
Accepted *on*, *off* (see *onoffkey*)
Example `task.putintparam(iparam.write_task_inc_sol, onoffkey.on)`
Generic name MSK_IPAR_WRITE_TASK_INC_SOL
Groups *Data input/output*

`iparam.write_xml_mode`
Controls if linear coefficients should be written by row or column when writing in the XML file format.

Default *row*
Accepted *row*, *col* (see *xmlwriteroutputtype*)
Example `task.putintparam(iparam.write_xml_mode, xmlwriteroutputtype.row)`
Generic name MSK_IPAR_WRITE_XML_MODE
Groups *Data input/output*

15.7.3 String parameters

`sparam`
The enumeration type containing all string parameters.

`sparam.bas_sol_file_name`
Name of the `bas` solution file.

Accepted Any valid file name.
Example `task.putstrparam(sparam.bas_sol_file_name, "somevalue")`
Generic name MSK_SPAR_BAS_SOL_FILE_NAME
Groups *Data input/output*, *Solution input/output*

`sparam.data_file_name`
Data are read and written to this file.

Accepted Any valid file name.
Example `task.putstrparam(sparam.data_file_name, "somevalue")`
Generic name MSK_SPAR_DATA_FILE_NAME
Groups *Data input/output*

`sparam.debug_file_name`
MOSEK debug file.

Accepted Any valid file name.
Example `task.putstrparam(sparam.debug_file_name, "somevalue")`
Generic name MSK_SPAR_DEBUG_FILE_NAME
Groups *Data input/output*

`sparam.int_sol_file_name`
Name of the `int` solution file.

Accepted Any valid file name.
Example `task.putstrparam(sparam.int_sol_file_name, "somevalue")`
Generic name MSK_SPAR_INT_SOL_FILE_NAME
Groups *Data input/output*, *Solution input/output*

`sparam.itr_sol_file_name`
Name of the `itr` solution file.

Accepted Any valid file name.
Example `task.putstrparam(sparam.itr_sol_file_name, "somevalue")`
Generic name MSK_SPAR_ITR_SOL_FILE_NAME
Groups *Data input/output*, *Solution input/output*

`sparam.mio_debug_string`
For internal debugging purposes.

Accepted Any valid string.
Example `task.putstrparam(sparam.mio_debug_string, "somevalue")`
Generic name MSK_SPAR_MIO_DEBUG_STRING
Groups *Data input/output*

`sparam.param_comment_sign`
Only the first character in this string is used. It is considered as a start of comment sign in the **MOSEK** parameter file. Spaces are ignored in the string.

Default
%%
Accepted Any valid string.
Example `task.putstrparam(sparam.param_comment_sign, "%%")`

Generic name MSK_SPAR_PARAM_COMMENT_SIGN
Groups *Data input/output*

`sparam.param_read_file_name`
Modifications to the parameter database is read from this file.

Accepted Any valid file name.
Example `task.putstrparam(sparam.param_read_file_name, "somevalue")`
Generic name MSK_SPAR_PARAM_READ_FILE_NAME
Groups *Data input/output*

`sparam.param_write_file_name`
The parameter database is written to this file.

Accepted Any valid file name.
Example `task.putstrparam(sparam.param_write_file_name, "somevalue")`
Generic name MSK_SPAR_PARAM_WRITE_FILE_NAME
Groups *Data input/output*

`sparam.read_mps_bou_name`
Name of the BOUNDS vector used. An empty name means that the first BOUNDS vector is used.

Accepted Any valid MPS name.
Example `task.putstrparam(sparam.read_mps_bou_name, "somevalue")`
Generic name MSK_SPAR_READ_MPS_BOU_NAME
Groups *Data input/output*

`sparam.read_mps_obj_name`
Name of the free constraint used as objective function. An empty name means that the first constraint is used as objective function.

Accepted Any valid MPS name.
Example `task.putstrparam(sparam.read_mps_obj_name, "somevalue")`
Generic name MSK_SPAR_READ_MPS_OBJ_NAME
Groups *Data input/output*

`sparam.read_mps_ran_name`
Name of the RANGE vector used. An empty name means that the first RANGE vector is used.

Accepted Any valid MPS name.
Example `task.putstrparam(sparam.read_mps_ran_name, "somevalue")`
Generic name MSK_SPAR_READ_MPS_RAN_NAME
Groups *Data input/output*

`sparam.read_mps_rhs_name`
Name of the RHS used. An empty name means that the first RHS vector is used.

Accepted Any valid MPS name.
Example `task.putstrparam(sparam.read_mps_rhs_name, "somevalue")`
Generic name MSK_SPAR_READ_MPS_RHS_NAME
Groups *Data input/output*

`sparam.remote_access_token`
An access token used to submit tasks to a remote **MOSEK** server. An access token is a random 32-byte string encoded in base64, i.e. it is a 44 character ASCII string.

Accepted Any valid string.
Example `task.putstrparam(sparam.remote_access_token, "somevalue")`
Generic name MSK_SPAR_REMOTE_ACCESS_TOKEN
Groups *Overall system*

`sparam.sensitivity_file_name`
If defined *Task.sensitivityreport* reads this file as a sensitivity analysis data file specifying the type of analysis to be done.

Accepted Any valid string.
Example `task.putstrparam(sparam.sensitivity_file_name, "somevalue")`
Generic name MSK_SPAR_SENSITIVITY_FILE_NAME
Groups *Data input/output*

`sparam.sensitivity_res_file_name`
If this is a nonempty string, then *Task.sensitivityreport* writes results to this file.

Accepted Any valid string.
Example `task.putstrparam(sparam.sensitivity_res_file_name, "somevalue")`
Generic name MSK_SPAR_SENSITIVITY_RES_FILE_NAME
Groups *Data input/output*

`sparam.sol_filter_xc_low`
A filter used to determine which constraints should be listed in the solution file. A value of 0.5 means that all constraints having `xc[i]>0.5` should be listed, whereas `+0.5` means that all constraints having `xc[i]>=blc[i]+0.5` should be listed. An empty filter means that no filter is applied.

Accepted Any valid filter.
Example `task.putstrparam(sparam.sol_filter_xc_low, "somevalue")`
Generic name MSK_SPAR_SOL_FILTER_XC_LOW
Groups *Data input/output*, *Solution input/output*

`sparam.sol_filter_xc_upr`
A filter used to determine which constraints should be listed in the solution file. A value of `0.5` means that all constraints having `xc[i]<0.5` should be listed, whereas `-0.5` means all constraints having `xc[i]<=buc[i]-0.5` should be listed. An empty filter means that no filter is applied.

Accepted Any valid filter.
Example `task.putstrparam(sparam.sol_filter_xc_upr, "somevalue")`
Generic name MSK_SPAR_SOL_FILTER_XC_UPR
Groups *Data input/output*, *Solution input/output*

`sparam.sol_filter_xx_low`
A filter used to determine which variables should be listed in the solution file. A value of "0.5" means that all constraints having `xx[j]>=0.5` should be listed, whereas "+0.5" means that all constraints having `xx[j]>=blx[j]+0.5` should be listed. An empty filter means no filter is applied.

Accepted Any valid filter.
Example `task.putstrparam(sparam.sol_filter_xx_low, "somevalue")`
Generic name MSK_SPAR_SOL_FILTER_XX_LOW
Groups *Data input/output*, *Solution input/output*

`sparam.sol_filter_xx_upr`
A filter used to determine which variables should be listed in the solution file. A value of "0.5" means that all constraints having `xx[j]<0.5` should be printed, whereas "-0.5" means all constraints having `xx[j]<=bux[j]-0.5` should be listed. An empty filter means no filter is applied.

Accepted Any valid file name.
Example `task.putstrparam(sparam.sol_filter_xx_upr, "somevalue")`
Generic name MSK_SPAR_SOL_FILTER_XX_UPR
Groups *Data input/output*, *Solution input/output*

`sparam.stat_file_name`
Statistics file name.

Accepted Any valid file name.
Example `task.putstrparam(sparam.stat_file_name, "somevalue")`
Generic name MSK_SPAR_STAT_FILE_NAME
Groups *Data input/output*

`sparam.stat_key`
Key used when writing the summary file.

Accepted Any valid string.
Example `task.putstrparam(sparam.stat_key, "somevalue")`
Generic name MSK_SPAR_STAT_KEY
Groups *Data input/output*

`sparam.stat_name`
Name used when writing the statistics file.

Accepted Any valid XML string.
Example `task.putstrparam(sparam.stat_name, "somevalue")`
Generic name MSK_SPAR_STAT_NAME
Groups *Data input/output*

`sparam.write_lp_gen_var_name`
Sometimes when an LP file is written additional variables must be inserted. They will have the prefix denoted by this parameter.

Default xmskgen
Accepted Any valid string.
Example `task.putstrparam(sparam.write_lp_gen_var_name, "xmskgen")`
Generic name MSK_SPAR_WRITE_LP_GEN_VAR_NAME
Groups *Data input/output*

15.8 Response codes

Response codes include:

- *Termination codes*
- *Warnings*
- *Errors*

The numerical code (in brackets) identifies the response in error messages and in the log output.

`rescode`
The enumeration type containing all response codes.

15.8.1 Termination

`rescode.ok (0)`
No error occurred.

`rescode.trm_max_iterations (10000)`
The optimizer terminated at the maximum number of iterations.

`rescode.trm_max_time (10001)`
The optimizer terminated at the maximum amount of time.

`rescode.trm_objective_range (10002)`
The optimizer terminated with an objective value outside the objective range.

`rescode.trm_mio_num_relaxs (10008)`
The mixed-integer optimizer terminated as the maximum number of relaxations was reached.

`rescode.trm_mio_num_branches (10009)`
The mixed-integer optimizer terminated as the maximum number of branches was reached.

rescode.trm_num_max_num_int_solutions (10015)
: The mixed-integer optimizer terminated as the maximum number of feasible solutions was reached.

rescode.trm_stall (10006)
: The optimizer is terminated due to slow progress.

Stalling means that numerical problems prevent the optimizer from making reasonable progress and that it makes no sense to continue. In many cases this happens if the problem is badly scaled or otherwise ill-conditioned. There is no guarantee that the solution will be feasible or optimal. However, often stalling happens near the optimum, and the returned solution may be of good quality. Therefore, it is recommended to check the status of the solution. If the solution status is optimal the solution is most likely good enough for most practical purposes.

Please note that if a linear optimization problem is solved using the interior-point optimizer with basis identification turned on, the returned basic solution likely to have high accuracy, even though the optimizer stalled.

Some common causes of stalling are a) badly scaled models, b) near feasible or near infeasible problems.

rescode.trm_user_callback (10007)
: The optimizer terminated due to the return of the user-defined callback function.

rescode.trm_max_num_setbacks (10020)
: The optimizer terminated as the maximum number of set-backs was reached. This indicates serious numerical problems and a possibly badly formulated problem.

rescode.trm_numerical_problem (10025)
: The optimizer terminated due to numerical problems.

rescode.trm_internal (10030)
: The optimizer terminated due to some internal reason. Please contact **MOSEK** support.

rescode.trm_internal_stop (10031)
: The optimizer terminated for internal reasons. Please contact **MOSEK** support.

15.8.2 Warnings

rescode.wrn_open_param_file (50)
: The parameter file could not be opened.

rescode.wrn_large_bound (51)
: A numerically large bound value is specified.

rescode.wrn_large_lo_bound (52)
: A numerically large lower bound value is specified.

rescode.wrn_large_up_bound (53)
: A numerically large upper bound value is specified.

rescode.wrn_large_con_fx (54)
: An equality constraint is fixed to a numerically large value. This can cause numerical problems.

rescode.wrn_large_cj (57)
: A numerically large value is specified for one c_j.

rescode.wrn_large_aij (62)
: A numerically large value is specified for an $a_{i,j}$ element in A. The parameter *dparam.data_tol_aij_large* controls when an $a_{i,j}$ is considered large.

rescode.wrn_zero_aij (63)
: One or more zero elements are specified in A.

rescode.wrn_name_max_len (65)
: A name is longer than the buffer that is supposed to hold it.

rescode.wrn_spar_max_len (66)
: A value for a string parameter is longer than the buffer that is supposed to hold it.

rescode.wrn_mps_split_rhs_vector (70)
: An RHS vector is split into several nonadjacent parts in an MPS file.

rescode.wrn_mps_split_ran_vector (71)
: A RANGE vector is split into several nonadjacent parts in an MPS file.

rescode.wrn_mps_split_bou_vector (72)
: A BOUNDS vector is split into several nonadjacent parts in an MPS file.

rescode.wrn_lp_old_quad_format (80)

Missing '/2' after quadratic expressions in bound or objective.

rescode.wrn_lp_drop_variable (85)

Ignored a variable because the variable was not previously defined. Usually this implies that a variable appears in the bound section but not in the objective or the constraints.

rescode.wrn_nz_in_upr_tri (200)

Non-zero elements specified in the upper triangle of a matrix were ignored.

rescode.wrn_dropped_nz_qobj (201)

One or more non-zero elements were dropped in the Q matrix in the objective.

rescode.wrn_ignore_integer (250)

Ignored integer constraints.

rescode.wrn_no_global_optimizer (251)

No global optimizer is available.

rescode.wrn_mio_infeasible_final (270)

The final mixed-integer problem with all the integer variables fixed at their optimal values is infeasible.

rescode.wrn_sol_filter (300)

Invalid solution filter is specified.

rescode.wrn_undef_sol_file_name (350)

Undefined name occurred in a solution.

rescode.wrn_sol_file_ignored_con (351)

One or more lines in the constraint section were ignored when reading a solution file.

rescode.wrn_sol_file_ignored_var (352)

One or more lines in the variable section were ignored when reading a solution file.

rescode.wrn_too_few_basis_vars (400)

An incomplete basis has been specified. Too few basis variables are specified.

rescode.wrn_too_many_basis_vars (405)

A basis with too many variables has been specified.

rescode.wrn_license_expire (500)

The license expires.

rescode.wrn_license_server (501)

The license server is not responding.

rescode.wrn_empty_name (502)

A variable or constraint name is empty. The output file may be invalid.

rescode.wrn_using_generic_names (503)

Generic names are used because a name is not valid. For instance when writing an LP file the names must not contain blanks or start with a digit.

rescode.wrn_license_feature_expire (505)

The license expires.

rescode.wrn_param_name_dou (510)

The parameter name is not recognized as a double parameter.

rescode.wrn_param_name_int (511)

The parameter name is not recognized as a integer parameter.

rescode.wrn_param_name_str (512)

The parameter name is not recognized as a string parameter.

rescode.wrn_param_str_value (515)

The string is not recognized as a symbolic value for the parameter.

rescode.wrn_param_ignored_cmio (516)

A parameter was ignored by the conic mixed integer optimizer.

rescode.wrn_zeros_in_sparse_row (705)

One or more (near) zero elements are specified in a sparse row of a matrix. Since, it is redundant to specify zero elements then it may indicate an error.

rescode.wrn_zeros_in_sparse_col (710)

One or more (near) zero elements are specified in a sparse column of a matrix. It is redundant to specify zero elements. Hence, it may indicate an error.

rescode.wrn_incomplete_linear_dependency_check (800)

The linear dependency check(s) is incomplete. Normally this is not an important warning unless the optimization problem has been formulated with linear dependencies. Linear dependencies may prevent **MOSEK** from solving the problem.

`rescode.wrn_eliminator_space (801)`
The eliminator is skipped at least once due to lack of space.

`rescode.wrn_presolve_outofspace (802)`
The presolve is incomplete due to lack of space.

`rescode.wrn_write_changed_names (803)`
Some names were changed because they were invalid for the output file format.

`rescode.wrn_write_discarded_cfix (804)`
The fixed objective term could not be converted to a variable and was discarded in the output file.

`rescode.wrn_duplicate_constraint_names (850)`
Two constraint names are identical.

`rescode.wrn_duplicate_variable_names (851)`
Two variable names are identical.

`rescode.wrn_duplicate_barvariable_names (852)`
Two barvariable names are identical.

`rescode.wrn_duplicate_cone_names (853)`
Two cone names are identical.

`rescode.wrn_ana_large_bounds (900)`
This warning is issued by the problem analyzer, if one or more constraint or variable bounds are very large. One should consider omitting these bounds entirely by setting them to +inf or -inf.

`rescode.wrn_ana_c_zero (901)`
This warning is issued by the problem analyzer, if the coefficients in the linear part of the objective are all zero.

`rescode.wrn_ana_empty_cols (902)`
This warning is issued by the problem analyzer, if columns, in which all coefficients are zero, are found.

`rescode.wrn_ana_close_bounds (903)`
This warning is issued by problem analyzer, if ranged constraints or variables with very close upper and lower bounds are detected. One should consider treating such constraints as equalities and such variables as constants.

`rescode.wrn_ana_almost_int_bounds (904)`
This warning is issued by the problem analyzer if a constraint is bound nearly integral.

`rescode.wrn_quad_cones_with_root_fixed_at_zero (930)`
For at least one quadratic cone the root is fixed at (nearly) zero. This may cause problems such as a very large dual solution. Therefore, it is recommended to remove such cones before optimizing the problem, or to fix all the variables in the cone to 0.

`rescode.wrn_rquad_cones_with_root_fixed_at_zero (931)`
For at least one rotated quadratic cone at least one of the root variables are fixed at (nearly) zero. This may cause problems such as a very large dual solution. Therefore, it is recommended to remove such cones before optimizing the problem, or to fix all the variables in the cone to 0.

`rescode.wrn_exp_cones_with_variables_fixed_at_zero (932)`
For at least one exponential cone $x \geq y \exp(z/y)$ either the variable x or y is fixed at (nearly) zero. This may cause problems such as a very large dual solution. Therefore, it is recommended to remove such cones before optimizing the problem, or to fix all the variables in the cone to 0.

`rescode.wrn_pow_cones_with_root_fixed_at_zero (933)`
For at least one power cone at least one of the root variables are fixed at (nearly) zero. This may cause problems such as a very large dual solution. Therefore, it is recommended to remove such cones before optimizing the problem, or to fix all the variables in the cone to 0.

`rescode.wrn_no_dualizer (950)`
No automatic dualizer is available for the specified problem. The primal problem is solved.

`rescode.wrn_sym_mat_large (960)`
A numerically large value is specified for an $e_{i,j}$ element in E. The parameter *dparam.data_sym_mat_tol_large* controls when an $e_{i,j}$ is considered large.

15.8.3 Errors

`rescode.err_license (1000)`
: Invalid license.

`rescode.err_license_expired (1001)`
: The license has expired.

`rescode.err_license_version (1002)`
: The license is valid for another version of **MOSEK**.

`rescode.err_size_license (1005)`
: The problem is bigger than the license.

`rescode.err_prob_license (1006)`
: The software is not licensed to solve the problem.

`rescode.err_file_license (1007)`
: Invalid license file.

`rescode.err_missing_license_file (1008)`
: **MOSEK** cannot find license file or a token server. See the **MOSEK** licensing manual for details.

`rescode.err_size_license_con (1010)`
: The problem has too many constraints to be solved with the available license.

`rescode.err_size_license_var (1011)`
: The problem has too many variables to be solved with the available license.

`rescode.err_size_license_intvar (1012)`
: The problem contains too many integer variables to be solved with the available license.

`rescode.err_optimizer_license (1013)`
: The optimizer required is not licensed.

`rescode.err_flexlm (1014)`
: The FLEXlm license manager reported an error.

`rescode.err_license_server (1015)`
: The license server is not responding.

`rescode.err_license_max (1016)`
: Maximum number of licenses is reached.

`rescode.err_license_moseklm_daemon (1017)`
: The MOSEKLM license manager daemon is not up and running.

`rescode.err_license_feature (1018)`
: A requested feature is not available in the license file(s). Most likely due to an incorrect license system setup.

`rescode.err_platform_not_licensed (1019)`
: A requested license feature is not available for the required platform.

`rescode.err_license_cannot_allocate (1020)`
: The license system cannot allocate the memory required.

`rescode.err_license_cannot_connect (1021)`
: **MOSEK** cannot connect to the license server. Most likely the license server is not up and running.

`rescode.err_license_invalid_hostid (1025)`
: The host ID specified in the license file does not match the host ID of the computer.

`rescode.err_license_server_version (1026)`
: The version specified in the checkout request is greater than the highest version number the daemon supports.

`rescode.err_license_no_server_support (1027)`
: The license server does not support the requested feature. Possible reasons for this error include:

  - The feature has expired.
  - The feature's start date is later than today's date.
  - The version requested is higher than feature's the highest supported version.
  - A corrupted license file.

  Try restarting the license and inspect the license server debug file, usually called `lmgrd.log`.

`rescode.err_license_no_server_line (1028)`
: There is no `SERVER` line in the license file. All non-zero license count features need at least one `SERVER` line.

`rescode.err_older_dll (1035)`
: The dynamic link library is older than the specified version.

`rescode.err_newer_dll (1036)`
: The dynamic link library is newer than the specified version.

`rescode.err_link_file_dll (1040)`
: A file cannot be linked to a stream in the DLL version.

`rescode.err_thread_mutex_init (1045)`
: Could not initialize a mutex.

`rescode.err_thread_mutex_lock (1046)`
: Could not lock a mutex.

`rescode.err_thread_mutex_unlock (1047)`
: Could not unlock a mutex.

`rescode.err_thread_create (1048)`
: Could not create a thread. This error may occur if a large number of environments are created and not deleted again. In any case it is a good practice to minimize the number of environments created.

`rescode.err_thread_cond_init (1049)`
: Could not initialize a condition.

`rescode.err_unknown (1050)`
: Unknown error.

`rescode.err_space (1051)`
: Out of space.

`rescode.err_file_open (1052)`
: Error while opening a file.

`rescode.err_file_read (1053)`
: File read error.

`rescode.err_file_write (1054)`
: File write error.

`rescode.err_data_file_ext (1055)`
: The data file format cannot be determined from the file name.

`rescode.err_invalid_file_name (1056)`
: An invalid file name has been specified.

`rescode.err_invalid_sol_file_name (1057)`
: An invalid file name has been specified.

`rescode.err_end_of_file (1059)`
: End of file reached.

`rescode.err_null_env (1060)`
: `env` is a NULL pointer.

`rescode.err_null_task (1061)`
: `task` is a NULL pointer.

`rescode.err_invalid_stream (1062)`
: An invalid stream is referenced.

`rescode.err_no_init_env (1063)`
: `env` is not initialized.

`rescode.err_invalid_task (1064)`
: The `task` is invalid.

`rescode.err_null_pointer (1065)`
: An argument to a function is unexpectedly a NULL pointer.

`rescode.err_living_tasks (1066)`
: All tasks associated with an enviroment must be deleted before the environment is deleted. There are still some undeleted tasks.

`rescode.err_blank_name (1070)`
: An all blank name has been specified.

`rescode.err_dup_name (1071)`
: The same name was used multiple times for the same problem item type.

`rescode.err_format_string (1072)`
: The name format string is invalid.

rescode.err_invalid_obj_name (1075)
An invalid objective name is specified.

rescode.err_invalid_con_name (1076)
An invalid constraint name is used.

rescode.err_invalid_var_name (1077)
An invalid variable name is used.

rescode.err_invalid_cone_name (1078)
An invalid cone name is used.

rescode.err_invalid_barvar_name (1079)
An invalid symmetric matrix variable name is used.

rescode.err_space_leaking (1080)
MOSEK is leaking memory. This can be due to either an incorrect use of **MOSEK** or a bug.

rescode.err_space_no_info (1081)
No available information about the space usage.

rescode.err_read_format (1090)
The specified format cannot be read.

rescode.err_mps_file (1100)
An error occurred while reading an MPS file.

rescode.err_mps_inv_field (1101)
A field in the MPS file is invalid. Probably it is too wide.

rescode.err_mps_inv_marker (1102)
An invalid marker has been specified in the MPS file.

rescode.err_mps_null_con_name (1103)
An empty constraint name is used in an MPS file.

rescode.err_mps_null_var_name (1104)
An empty variable name is used in an MPS file.

rescode.err_mps_undef_con_name (1105)
An undefined constraint name occurred in an MPS file.

rescode.err_mps_undef_var_name (1106)
An undefined variable name occurred in an MPS file.

rescode.err_mps_inv_con_key (1107)
An invalid constraint key occurred in an MPS file.

rescode.err_mps_inv_bound_key (1108)
An invalid bound key occurred in an MPS file.

rescode.err_mps_inv_sec_name (1109)
An invalid section name occurred in an MPS file.

rescode.err_mps_no_objective (1110)
No objective is defined in an MPS file.

rescode.err_mps_splitted_var (1111)
All elements in a column of the A matrix must be specified consecutively. Hence, it is illegal to specify non-zero elements in A for variable 1, then for variable 2 and then variable 1 again.

rescode.err_mps_mul_con_name (1112)
A constraint name was specified multiple times in the ROWS section.

rescode.err_mps_mul_qsec (1113)
Multiple QSECTIONs are specified for a constraint in the MPS data file.

rescode.err_mps_mul_qobj (1114)
The Q term in the objective is specified multiple times in the MPS data file.

rescode.err_mps_inv_sec_order (1115)
The sections in the MPS data file are not in the correct order.

rescode.err_mps_mul_csec (1116)
Multiple CSECTIONs are given the same name.

rescode.err_mps_cone_type (1117)
Invalid cone type specified in a CSECTION.

rescode.err_mps_cone_overlap (1118)
A variable is specified to be a member of several cones.

rescode.err_mps_cone_repeat (1119)
A variable is repeated within the CSECTION.

rescode.err_mps_non_symmetric_q (1120)
A non symmetric matrix has been speciefied.

rescode.err_mps_duplicate_q_element (1121)
Duplicate elements is specfied in a Q matrix.

rescode.err_mps_invalid_objsense (1122)
An invalid objective sense is specified.

rescode.err_mps_tab_in_field2 (1125)
A tab char occurred in field 2.

rescode.err_mps_tab_in_field3 (1126)
A tab char occurred in field 3.

rescode.err_mps_tab_in_field5 (1127)
A tab char occurred in field 5.

rescode.err_mps_invalid_obj_name (1128)
An invalid objective name is specified.

rescode.err_lp_incompatible (1150)
The problem cannot be written to an LP formatted file.

rescode.err_lp_empty (1151)
The problem cannot be written to an LP formatted file.

rescode.err_lp_dup_slack_name (1152)
The name of the slack variable added to a ranged constraint already exists.

rescode.err_write_mps_invalid_name (1153)
An invalid name is created while writing an MPS file. Usually this will make the MPS file unreadable.

rescode.err_lp_invalid_var_name (1154)
A variable name is invalid when used in an LP formatted file.

rescode.err_lp_free_constraint (1155)
Free constraints cannot be written in LP file format.

rescode.err_write_opf_invalid_var_name (1156)
Empty variable names cannot be written to OPF files.

rescode.err_lp_file_format (1157)
Syntax error in an LP file.

rescode.err_write_lp_format (1158)
Problem cannot be written as an LP file.

rescode.err_read_lp_missing_end_tag (1159)
Syntax error in LP file. Possibly missing End tag.

rescode.err_lp_format (1160)
Syntax error in an LP file.

rescode.err_write_lp_non_unique_name (1161)
An auto-generated name is not unique.

rescode.err_read_lp_nonexisting_name (1162)
A variable never occurred in objective or constraints.

rescode.err_lp_write_conic_problem (1163)
The problem contains cones that cannot be written to an LP formatted file.

rescode.err_lp_write_geco_problem (1164)
The problem contains general convex terms that cannot be written to an LP formatted file.

rescode.err_writing_file (1166)
An error occurred while writing file

rescode.err_ptf_format (1167)
Syntax error in an PTF file

rescode.err_opf_format (1168)
Syntax error in an OPF file

rescode.err_opf_new_variable (1169)
Introducing new variables is now allowed. When a [variables] section is present, it is not allowed to introduce new variables later in the problem.

rescode.err_invalid_name_in_sol_file (1170)
An invalid name occurred in a solution file.

rescode.err_lp_invalid_con_name (1171)
A constraint name is invalid when used in an LP formatted file.

rescode.err_opf_premature_eof (1172)
Premature end of file in an OPF file.

rescode.err_json_syntax (1175)
Syntax error in an JSON data

rescode.err_json_string (1176)
Error in JSON string.

rescode.err_json_number_overflow (1177)
Invalid number entry - wrong type or value overflow.

rescode.err_json_format (1178)
Error in an JSON Task file

rescode.err_json_data (1179)
Inconsistent data in JSON Task file

rescode.err_json_missing_data (1180)
Missing data section in JSON task file.

rescode.err_argument_lenneq (1197)
Incorrect length of arguments.

rescode.err_argument_type (1198)
Incorrect argument type.

rescode.err_num_arguments (1199)
Incorrect number of function arguments.

rescode.err_in_argument (1200)
A function argument is incorrect.

rescode.err_argument_dimension (1201)
A function argument is of incorrect dimension.

rescode.err_shape_is_too_large (1202)
The size of the n-dimensional shape is too large.

rescode.err_index_is_too_small (1203)
An index in an argument is too small.

rescode.err_index_is_too_large (1204)
An index in an argument is too large.

rescode.err_param_name (1205)
The parameter name is not correct.

rescode.err_param_name_dou (1206)
The parameter name is not correct for a double parameter.

rescode.err_param_name_int (1207)
The parameter name is not correct for an integer parameter.

rescode.err_param_name_str (1208)
The parameter name is not correct for a string parameter.

rescode.err_param_index (1210)
Parameter index is out of range.

rescode.err_param_is_too_large (1215)
The parameter value is too large.

rescode.err_param_is_too_small (1216)
The parameter value is too small.

rescode.err_param_value_str (1217)
The parameter value string is incorrect.

rescode.err_param_type (1218)
The parameter type is invalid.

rescode.err_inf_dou_index (1219)
A double information index is out of range for the specified type.

rescode.err_inf_int_index (1220)
An integer information index is out of range for the specified type.

rescode.err_index_arr_is_too_small (1221)
An index in an array argument is too small.

rescode.err_index_arr_is_too_large (1222)
An index in an array argument is too large.

rescode.err_inf_lint_index (1225)
A long integer information index is out of range for the specified type.

rescode.err_arg_is_too_small (1226)
The value of a argument is too small.
rescode.err_arg_is_too_large (1227)
The value of a argument is too large.
rescode.err_invalid_whichsol (1228)
whichsol is invalid.
rescode.err_inf_dou_name (1230)
A double information name is invalid.
rescode.err_inf_int_name (1231)
An integer information name is invalid.
rescode.err_inf_type (1232)
The information type is invalid.
rescode.err_inf_lint_name (1234)
A long integer information name is invalid.
rescode.err_index (1235)
An index is out of range.
rescode.err_whichsol (1236)
The solution defined by whichsol does not exists.
rescode.err_solitem (1237)
The solution item number solitem is invalid. Please note that *solitem.snx* is invalid for the basic solution.
rescode.err_whichitem_not_allowed (1238)
whichitem is unacceptable.
rescode.err_maxnumcon (1240)
The maximum number of constraints specified is smaller than the number of constraints in the task.
rescode.err_maxnumvar (1241)
The maximum number of variables specified is smaller than the number of variables in the task.
rescode.err_maxnumbarvar (1242)
The maximum number of semidefinite variables specified is smaller than the number of semidefinite variables in the task.
rescode.err_maxnumqnz (1243)
The maximum number of non-zeros specified for the Q matrices is smaller than the number of non-zeros in the current Q matrices.
rescode.err_too_small_max_num_nz (1245)
The maximum number of non-zeros specified is too small.
rescode.err_invalid_idx (1246)
A specified index is invalid.
rescode.err_invalid_max_num (1247)
A specified index is invalid.
rescode.err_numconlim (1250)
Maximum number of constraints limit is exceeded.
rescode.err_numvarlim (1251)
Maximum number of variables limit is exceeded.
rescode.err_too_small_maxnumanz (1252)
The maximum number of non-zeros specified for A is smaller than the number of non-zeros in the current A.
rescode.err_inv_aptre (1253)
aptre[j] is strictly smaller than aptrb[j] for some j.
rescode.err_mul_a_element (1254)
An element in A is defined multiple times.
rescode.err_inv_bk (1255)
Invalid bound key.
rescode.err_inv_bkc (1256)
Invalid bound key is specified for a constraint.
rescode.err_inv_bkx (1257)
An invalid bound key is specified for a variable.

rescode.err_inv_var_type (1258)
An invalid variable type is specified for a variable.
rescode.err_solver_probtype (1259)
Problem type does not match the chosen optimizer.
rescode.err_objective_range (1260)
Empty objective range.
rescode.err_undef_solution (1265)
MOSEK has the following solution types:

- an interior-point solution,
- a basic solution,
- and an integer solution.

Each optimizer may set one or more of these solutions; e.g by default a successful optimization with the interior-point optimizer defines the interior-point solution and, for linear problems, also the basic solution. This error occurs when asking for a solution or for information about a solution that is not defined.
rescode.err_basis (1266)
An invalid basis is specified. Either too many or too few basis variables are specified.
rescode.err_inv_skc (1267)
Invalid value in skc.
rescode.err_inv_skx (1268)
Invalid value in skx.
rescode.err_inv_skn (1274)
Invalid value in skn.
rescode.err_inv_sk_str (1269)
Invalid status key string encountered.
rescode.err_inv_sk (1270)
Invalid status key code.
rescode.err_inv_cone_type_str (1271)
Invalid cone type string encountered.
rescode.err_inv_cone_type (1272)
Invalid cone type code is encountered.
rescode.err_invalid_surplus (1275)
Invalid surplus.
rescode.err_inv_name_item (1280)
An invalid name item code is used.
rescode.err_pro_item (1281)
An invalid problem is used.
rescode.err_invalid_format_type (1283)
Invalid format type.
rescode.err_firsti (1285)
Invalid firsti.
rescode.err_lasti (1286)
Invalid lasti.
rescode.err_firstj (1287)
Invalid firstj.
rescode.err_lastj (1288)
Invalid lastj.
rescode.err_max_len_is_too_small (1289)
A maximum length that is too small has been specified.
rescode.err_nonlinear_equality (1290)
The model contains a nonlinear equality which defines a nonconvex set.
rescode.err_nonconvex (1291)
The optimization problem is nonconvex.
rescode.err_nonlinear_ranged (1292)
Nonlinear constraints with finite lower and upper bound always define a nonconvex feasible set.

rescode.err_con_q_not_psd (1293)
The quadratic constraint matrix is not positive semidefinite as expected for a constraint with finite upper bound. This results in a nonconvex problem. The parameter *dparam.check_convexity_rel_tol* can be used to relax the convexity check.

rescode.err_con_q_not_nsd (1294)
The quadratic constraint matrix is not negative semidefinite as expected for a constraint with finite lower bound. This results in a nonconvex problem. The parameter *dparam.check_convexity_rel_tol* can be used to relax the convexity check.

rescode.err_obj_q_not_psd (1295)
The quadratic coefficient matrix in the objective is not positive semidefinite as expected for a minimization problem. The parameter *dparam.check_convexity_rel_tol* can be used to relax the convexity check.

rescode.err_obj_q_not_nsd (1296)
The quadratic coefficient matrix in the objective is not negative semidefinite as expected for a maximization problem. The parameter *dparam.check_convexity_rel_tol* can be used to relax the convexity check.

rescode.err_argument_perm_array (1299)
An invalid permutation array is specified.

rescode.err_cone_index (1300)
An index of a non-existing cone has been specified.

rescode.err_cone_size (1301)
A cone with incorrect number of members is specified.

rescode.err_cone_overlap (1302)
One or more of the variables in the cone to be added is already member of another cone. Now assume the variable is x_j then add a new variable say x_k and the constraint

$$x_j = x_k$$

and then let x_k be member of the cone to be appended.

rescode.err_cone_rep_var (1303)
A variable is included multiple times in the cone.

rescode.err_maxnumcone (1304)
The value specified for `maxnumcone` is too small.

rescode.err_cone_type (1305)
Invalid cone type specified.

rescode.err_cone_type_str (1306)
Invalid cone type specified.

rescode.err_cone_overlap_append (1307)
The cone to be appended has one variable which is already member of another cone.

rescode.err_remove_cone_variable (1310)
A variable cannot be removed because it will make a cone invalid.

rescode.err_appending_too_big_cone (1311)
Trying to append a too big cone.

rescode.err_cone_parameter (1320)
An invalid cone parameter.

rescode.err_sol_file_invalid_number (1350)
An invalid number is specified in a solution file.

rescode.err_huge_c (1375)
A huge value in absolute size is specified for one c_j.

rescode.err_huge_aij (1380)
A numerically huge value is specified for an $a_{i,j}$ element in A. The parameter *dparam.data_tol_aij_huge* controls when an $a_{i,j}$ is considered huge.

rescode.err_duplicate_aij (1385)
An element in the A matrix is specified twice.

rescode.err_lower_bound_is_a_nan (1390)
The lower bound specified is not a number (nan).

rescode.err_upper_bound_is_a_nan (1391)
The upper bound specified is not a number (nan).

rescode.err_infinite_bound (1400)
A numerically huge bound value is specified.

rescode.err_inv_qobj_subi (1401)
Invalid value in qosubi.

rescode.err_inv_qobj_subj (1402)
Invalid value in qosubj.

rescode.err_inv_qobj_val (1403)
Invalid value in qoval.

rescode.err_inv_qcon_subk (1404)
Invalid value in qcsubk.

rescode.err_inv_qcon_subi (1405)
Invalid value in qcsubi.

rescode.err_inv_qcon_subj (1406)
Invalid value in qcsubj.

rescode.err_inv_qcon_val (1407)
Invalid value in qcval.

rescode.err_qcon_subi_too_small (1408)
Invalid value in qcsubi.

rescode.err_qcon_subi_too_large (1409)
Invalid value in qcsubi.

rescode.err_qobj_upper_triangle (1415)
An element in the upper triangle of Q^o is specified. Only elements in the lower triangle should be specified.

rescode.err_qcon_upper_triangle (1417)
An element in the upper triangle of a Q^k is specified. Only elements in the lower triangle should be specified.

rescode.err_fixed_bound_values (1420)
A fixed constraint/variable has been specified using the bound keys but the numerical value of the lower and upper bound is different.

rescode.err_too_small_a_truncation_value (1421)
A too small value for the A trucation value is specified.

rescode.err_invalid_objective_sense (1445)
An invalid objective sense is specified.

rescode.err_undefined_objective_sense (1446)
The objective sense has not been specified before the optimization.

rescode.err_y_is_undefined (1449)
The solution item y is undefined.

rescode.err_nan_in_double_data (1450)
An invalid floating point value was used in some double data.

rescode.err_nan_in_blc (1461)
l^c contains an invalid floating point value, i.e. a NaN.

rescode.err_nan_in_buc (1462)
u^c contains an invalid floating point value, i.e. a NaN.

rescode.err_nan_in_c (1470)
c contains an invalid floating point value, i.e. a NaN.

rescode.err_nan_in_blx (1471)
l^x contains an invalid floating point value, i.e. a NaN.

rescode.err_nan_in_bux (1472)
u^x contains an invalid floating point value, i.e. a NaN.

rescode.err_invalid_aij (1473)
$a_{i,j}$ contains an invalid floating point value, i.e. a NaN or an infinite value.

rescode.err_sym_mat_invalid (1480)
A symmetric matrix contains an invalid floating point value, i.e. a NaN or an infinite value.

rescode.err_sym_mat_huge (1482)
A symmetric matrix contains a huge value in absolute size. The parameter *dparam.data_sym_mat_tol_huge* controls when an $e_{i,j}$ is considered huge.

rescode.err_inv_problem (1500)
Invalid problem type. Probably a nonconvex problem has been specified.

rescode.err_mixed_conic_and_nl (1501)
The problem contains nonlinear terms conic constraints. The requested operation cannot be applied to this type of problem.

rescode.err_global_inv_conic_problem (1503)
The global optimizer can only be applied to problems without semidefinite variables.

rescode.err_inv_optimizer (1550)
An invalid optimizer has been chosen for the problem.

rescode.err_mio_no_optimizer (1551)
No optimizer is available for the current class of integer optimization problems.

rescode.err_no_optimizer_var_type (1552)
No optimizer is available for this class of optimization problems.

rescode.err_final_solution (1560)
An error occurred during the solution finalization.

rescode.err_first (1570)
Invalid `first`.

rescode.err_last (1571)
Invalid index `last`. A given index was out of expected range.

rescode.err_slice_size (1572)
Invalid slice size specified.

rescode.err_negative_surplus (1573)
Negative surplus.

rescode.err_negative_append (1578)
Cannot append a negative number.

rescode.err_postsolve (1580)
An error occurred during the postsolve. Please contact **MOSEK** support.

rescode.err_overflow (1590)
A computation produced an overflow i.e. a very large number.

rescode.err_no_basis_sol (1600)
No basic solution is defined.

rescode.err_basis_factor (1610)
The factorization of the basis is invalid.

rescode.err_basis_singular (1615)
The basis is singular and hence cannot be factored.

rescode.err_factor (1650)
An error occurred while factorizing a matrix.

rescode.err_feasrepair_cannot_relax (1700)
An optimization problem cannot be relaxed.

rescode.err_feasrepair_solving_relaxed (1701)
The relaxed problem could not be solved to optimality. Please consult the log file for further details.

rescode.err_feasrepair_inconsistent_bound (1702)
The upper bound is less than the lower bound for a variable or a constraint. Please correct this before running the feasibility repair.

rescode.err_repair_invalid_problem (1710)
The feasibility repair does not support the specified problem type.

rescode.err_repair_optimization_failed (1711)
Computation the optimal relaxation failed. The cause may have been numerical problems.

rescode.err_name_max_len (1750)
A name is longer than the buffer that is supposed to hold it.

rescode.err_name_is_null (1760)
The name buffer is a NULL pointer.

rescode.err_invalid_compression (1800)
Invalid compression type.

rescode.err_invalid_iomode (1801)
Invalid io mode.

rescode.err_no_primal_infeas_cer (2000)
A certificate of primal infeasibility is not available.

rescode.err_no_dual_infeas_cer (2001)
A certificate of infeasibility is not available.

rescode.err_no_solution_in_callback (2500)
The required solution is not available.

rescode.err_inv_marki (2501)
Invalid value in marki.

rescode.err_inv_markj (2502)
Invalid value in markj.

rescode.err_inv_numi (2503)
Invalid numi.

rescode.err_inv_numj (2504)
Invalid numj.

rescode.err_task_incompatible (2560)
The Task file is incompatible with this platform. This results from reading a file on a 32 bit platform generated on a 64 bit platform.

rescode.err_task_invalid (2561)
The Task file is invalid.

rescode.err_task_write (2562)
Failed to write the task file.

rescode.err_lu_max_num_tries (2800)
Could not compute the LU factors of the matrix within the maximum number of allowed tries.

rescode.err_invalid_utf8 (2900)
An invalid UTF8 string is encountered.

rescode.err_invalid_wchar (2901)
An invalid `wchar` string is encountered.

rescode.err_no_dual_for_itg_sol (2950)
No dual information is available for the integer solution.

rescode.err_no_snx_for_bas_sol (2953)
s_n^x is not available for the basis solution.

rescode.err_internal (3000)
An internal error occurred. Please report this problem.

rescode.err_api_array_too_small (3001)
An input array was too short.

rescode.err_api_cb_connect (3002)
Failed to connect a callback object.

rescode.err_api_fatal_error (3005)
An internal error occurred in the API. Please report this problem.

rescode.err_api_internal (3999)
An internal fatal error occurred in an interface function.

rescode.err_sen_format (3050)
Syntax error in sensitivity analysis file.

rescode.err_sen_undef_name (3051)
An undefined name was encountered in the sensitivity analysis file.

rescode.err_sen_index_range (3052)
Index out of range in the sensitivity analysis file.

rescode.err_sen_bound_invalid_up (3053)
Analysis of upper bound requested for an index, where no upper bound exists.

rescode.err_sen_bound_invalid_lo (3054)
Analysis of lower bound requested for an index, where no lower bound exists.

rescode.err_sen_index_invalid (3055)
Invalid range given in the sensitivity file.

rescode.err_sen_invalid_regexp (3056)
Syntax error in regexp or regexp longer than 1024.

rescode.err_sen_solution_status (3057)
No optimal solution found to the original problem given for sensitivity analysis.

rescode.err_sen_numerical (3058)
Numerical difficulties encountered performing the sensitivity analysis.

rescode.err_sen_unhandled_problem_type (3080)
Sensitivity analysis cannot be performed for the specified problem. Sensitivity analysis is only possible for linear problems.

rescode.err_unb_step_size (3100)
A step size in an optimizer was unexpectedly unbounded. For instance, if the step-size becomes unbounded in phase 1 of the simplex algorithm then an error occurs. Normally this will happen only if the problem is badly formulated. Please contact **MOSEK** support if this error occurs.

rescode.err_identical_tasks (3101)
Some tasks related to this function call were identical. Unique tasks were expected.

rescode.err_ad_invalid_codelist (3102)
The code list data was invalid.

rescode.err_internal_test_failed (3500)
An internal unit test function failed.

rescode.err_xml_invalid_problem_type (3600)
The problem type is not supported by the XML format.

rescode.err_invalid_ampl_stub (3700)
Invalid AMPL stub.

rescode.err_int64_to_int32_cast (3800)
A 64 bit integer could not be cast to a 32 bit integer.

rescode.err_size_license_numcores (3900)
The computer contains more cpu cores than the license allows for.

rescode.err_infeas_undefined (3910)
The requested value is not defined for this solution type.

rescode.err_no_barx_for_solution (3915)
There is no $\overline{X}$ available for the solution specified. In particular note there are no $\overline{X}$ defined for the basic and integer solutions.

rescode.err_no_bars_for_solution (3916)
There is no $\bar{s}$ available for the solution specified. In particular note there are no $\bar{s}$ defined for the basic and integer solutions.

rescode.err_bar_var_dim (3920)
The dimension of a symmetric matrix variable has to be greater than 0.

rescode.err_sym_mat_invalid_row_index (3940)
A row index specified for sparse symmetric matrix is invalid.

rescode.err_sym_mat_invalid_col_index (3941)
A column index specified for sparse symmetric matrix is invalid.

rescode.err_sym_mat_not_lower_tringular (3942)
Only the lower triangular part of sparse symmetric matrix should be specified.

rescode.err_sym_mat_invalid_value (3943)
The numerical value specified in a sparse symmetric matrix is not a floating point value.

rescode.err_sym_mat_duplicate (3944)
A value in a symmetric matric as been specified more than once.

rescode.err_invalid_sym_mat_dim (3950)
A sparse symmetric matrix of invalid dimension is specified.

rescode.err_invalid_file_format_for_sym_mat (4000)
The file format does not support a problem with symmetric matrix variables.

rescode.err_invalid_file_format_for_cfix (4001)
The file format does not support a problem with nonzero fixed term in c.

rescode.err_invalid_file_format_for_ranged_constraints (4002)
The file format does not support a problem with ranged constraints.

rescode.err_invalid_file_format_for_free_constraints (4003)
The file format does not support a problem with free constraints.

rescode.err_invalid_file_format_for_cones (4005)
The file format does not support a problem with conic constraints.

rescode.err_invalid_file_format_for_nonlinear (4010)
The file format does not support a problem with nonlinear terms.

rescode.err_duplicate_constraint_names (4500)
Two constraint names are identical.

rescode.err_duplicate_variable_names (4501)
Two variable names are identical.

rescode.err_duplicate_barvariable_names (4502)
Two barvariable names are identical.

rescode.err_duplicate_cone_names (4503)
Two cone names are identical.

rescode.err_non_unique_array (5000)
An array does not contain unique elements.

rescode.err_argument_is_too_large (5005)
The value of a function argument is too large.

rescode.err_mio_internal (5010)
A fatal error occurred in the mixed integer optimizer. Please contact **MOSEK** support.

rescode.err_invalid_problem_type (6000)
An invalid problem type.

rescode.err_unhandled_solution_status (6010)
Unhandled solution status.

rescode.err_upper_triangle (6020)
An element in the upper triangle of a lower triangular matrix is specified.

rescode.err_lau_singular_matrix (7000)
A matrix is singular.

rescode.err_lau_not_positive_definite (7001)
A matrix is not positive definite.

rescode.err_lau_invalid_lower_triangular_matrix (7002)
An invalid lower triangular matrix.

rescode.err_lau_unknown (7005)
An unknown error.

rescode.err_lau_arg_m (7010)
Invalid argument m.

rescode.err_lau_arg_n (7011)
Invalid argument n.

rescode.err_lau_arg_k (7012)
Invalid argument k.

rescode.err_lau_arg_transa (7015)
Invalid argument transa.

rescode.err_lau_arg_transb (7016)
Invalid argument transb.

rescode.err_lau_arg_uplo (7017)
Invalid argument uplo.

rescode.err_lau_arg_trans (7018)
Invalid argument trans.

rescode.err_lau_invalid_sparse_symmetric_matrix (7019)
An invalid sparse symmetric matrix is specfified. Note only the lower triangular part with no duplicates is specifed.

rescode.err_cbf_parse (7100)
An error occurred while parsing an CBF file.

rescode.err_cbf_obj_sense (7101)
An invalid objective sense is specified.

rescode.err_cbf_no_variables (7102)
No variables are specified.

rescode.err_cbf_too_many_constraints (7103)
Too many constraints specified.

rescode.err_cbf_too_many_variables (7104)
Too many variables specified.

rescode.err_cbf_no_version_specified (7105)
No version specified.

rescode.err_cbf_syntax (7106)
Invalid syntax.

rescode.err_cbf_duplicate_obj (7107)
Duplicate OBJ keyword.

rescode.err_cbf_duplicate_con (7108)
Duplicate CON keyword.

rescode.err_cbf_duplicate_var (7109)
Duplicate VAR keyword.

rescode.err_cbf_duplicate_int (7110)
Duplicate INT keyword.

rescode.err_cbf_invalid_var_type (7111)
Invalid variable type.

rescode.err_cbf_invalid_con_type (7112)
Invalid constraint type.

rescode.err_cbf_invalid_domain_dimension (7113)
Invalid domain dimension.

rescode.err_cbf_duplicate_objacoord (7114)
Duplicate index in OBJCOORD.

rescode.err_cbf_duplicate_bcoord (7115)
Duplicate index in BCOORD.

rescode.err_cbf_duplicate_acoord (7116)
Duplicate index in ACOORD.

rescode.err_cbf_too_few_variables (7117)
Too few variables defined.

rescode.err_cbf_too_few_constraints (7118)
Too few constraints defined.

rescode.err_cbf_too_few_ints (7119)
Too few ints are specified.

rescode.err_cbf_too_many_ints (7120)
Too many ints are specified.

rescode.err_cbf_invalid_int_index (7121)
Invalid INT index.

rescode.err_cbf_unsupported (7122)
Unsupported feature is present.

rescode.err_cbf_duplicate_psdvar (7123)
Duplicate PSDVAR keyword.

rescode.err_cbf_invalid_psdvar_dimension (7124)
Invalid PSDVAR dimension.

rescode.err_cbf_too_few_psdvar (7125)
Too few variables defined.

rescode.err_cbf_invalid_exp_dimension (7126)
Invalid dimension of a exponential cone.

rescode.err_cbf_duplicate_pow_cones (7130)
Multiple POWCONES specified.

rescode.err_cbf_duplicate_pow_star_cones (7131)
Multiple POW*CONES specified.

rescode.err_cbf_invalid_power (7132)
Invalid power specified.

rescode.err_cbf_power_cone_is_too_long (7133)
Power cone is too long.

rescode.err_cbf_invalid_power_cone_index (7134)
Invalid power cone index.

rescode.err_cbf_invalid_power_star_cone_index (7135)
Invalid power star cone index.

rescode.err_cbf_unhandled_power_cone_type (7136)
An unhandled power cone type.

rescode.err_cbf_unhandled_power_star_cone_type (7137)
An unhandled power star cone type.

rescode.err_cbf_power_cone_mismatch (7138)
The power cone does not match with it definition.

rescode.err_cbf_power_star_cone_mismatch (7139)
The power star cone does not match with it definition.

rescode.err_cbf_invalid_number_of_cones (7740)
Invalid number of cones.

rescode.err_cbf_invalid_dimension_of_cones (7741)
Invalid dimension of cones.

rescode.err_mio_invalid_root_optimizer (7700)
An invalid root optimizer was selected for the problem type.

rescode.err_mio_invalid_node_optimizer (7701)
An invalid node optimizer was selected for the problem type.

rescode.err_toconic_constr_q_not_psd (7800)
The matrix defining the quadratic part of constraint is not positive semidefinite.

rescode.err_toconic_constraint_fx (7801)
The quadratic constraint is an equality, thus not convex.

rescode.err_toconic_constraint_ra (7802)
The quadratic constraint has finite lower and upper bound, and therefore it is not convex.

rescode.err_toconic_constr_not_conic (7803)
The constraint is not conic representable.

rescode.err_toconic_objective_not_psd (7804)
The matrix defining the quadratic part of the objective function is not positive semidefinite.

rescode.err_server_connect (8000)
Failed to connect to remote solver server. The server string or the port string were invalid, or the server did not accept connection.

rescode.err_server_protocol (8001)
Unexpected message or data from solver server.

rescode.err_server_status (8002)
Server returned non-ok HTTP status code

rescode.err_server_token (8003)
The job ID specified is incorrect or invalid

15.9 Enumerations

basindtype
Basis identification

basindtype.never
Never do basis identification.

basindtype.always
Basis identification is always performed even if the interior-point optimizer terminates abnormally.

basindtype.no_error
Basis identification is performed if the interior-point optimizer terminates without an error.

basindtype.if_feasible
Basis identification is not performed if the interior-point optimizer terminates with a problem status saying that the problem is primal or dual infeasible.

basindtype.reservered
Not currently in use.

boundkey
Bound keys

boundkey.lo
The constraint or variable has a finite lower bound and an infinite upper bound.

boundkey.up
The constraint or variable has an infinite lower bound and an finite upper bound.

boundkey.fx
The constraint or variable is fixed.

boundkey.fr
The constraint or variable is free.

boundkey.ra
The constraint or variable is ranged.

`mark`
: Mark

  `mark.lo`
  : The lower bound is selected for sensitivity analysis.

  `mark.up`
  : The upper bound is selected for sensitivity analysis.

`simdegen`
: Degeneracy strategies

  `simdegen.none`
  : The simplex optimizer should use no degeneration strategy.

  `simdegen.free`
  : The simplex optimizer chooses the degeneration strategy.

  `simdegen.aggressive`
  : The simplex optimizer should use an aggressive degeneration strategy.

  `simdegen.moderate`
  : The simplex optimizer should use a moderate degeneration strategy.

  `simdegen.minimum`
  : The simplex optimizer should use a minimum degeneration strategy.

`transpose`
: Transposed matrix.

  `transpose.no`
  : No transpose is applied.

  `transpose.yes`
  : A transpose is applied.

`uplo`
: Triangular part of a symmetric matrix.

  `uplo.lo`
  : Lower part.

  `uplo.up`
  : Upper part.

`simreform`
: Problem reformulation.

  `simreform.on`
  : Allow the simplex optimizer to reformulate the problem.

  `simreform.off`
  : Disallow the simplex optimizer to reformulate the problem.

  `simreform.free`
  : The simplex optimizer can choose freely.

  `simreform.aggressive`
  : The simplex optimizer should use an aggressive reformulation strategy.

`simdupvec`
: Exploit duplicate columns.

  `simdupvec.on`
  : Allow the simplex optimizer to exploit duplicated columns.

  `simdupvec.off`
  : Disallow the simplex optimizer to exploit duplicated columns.

  `simdupvec.free`
  : The simplex optimizer can choose freely.

`simhotstart`
: Hot-start type employed by the simplex optimizer

  `simhotstart.none`
  : The simplex optimizer performs a coldstart.

simhotstart.free
The simplex optimize chooses the hot-start type.

simhotstart.status_keys
Only the status keys of the constraints and variables are used to choose the type of hot-start.

intpnthotstart
Hot-start type employed by the interior-point optimizers.

intpnthotstart.none
The interior-point optimizer performs a coldstart.

intpnthotstart.primal
The interior-point optimizer exploits the primal solution only.

intpnthotstart.dual
The interior-point optimizer exploits the dual solution only.

intpnthotstart.primal_dual
The interior-point optimizer exploits both the primal and dual solution.

purify
Solution purification employed optimizer.

purify.none
The optimizer performs no solution purification.

purify.primal
The optimizer purifies the primal solution.

purify.dual
The optimizer purifies the dual solution.

purify.primal_dual
The optimizer purifies both the primal and dual solution.

purify.auto
TBD

callbackcode
Progress callback codes

callbackcode.begin_bi
The basis identification procedure has been started.

callbackcode.begin_conic
The callback function is called when the conic optimizer is started.

callbackcode.begin_dual_bi
The callback function is called from within the basis identification procedure when the dual phase is started.

callbackcode.begin_dual_sensitivity
Dual sensitivity analysis is started.

callbackcode.begin_dual_setup_bi
The callback function is called when the dual BI phase is started.

callbackcode.begin_dual_simplex
The callback function is called when the dual simplex optimizer started.

callbackcode.begin_dual_simplex_bi
The callback function is called from within the basis identification procedure when the dual simplex clean-up phase is started.

callbackcode.begin_full_convexity_check
Begin full convexity check.

callbackcode.begin_infeas_ana
The callback function is called when the infeasibility analyzer is started.

callbackcode.begin_intpnt
The callback function is called when the interior-point optimizer is started.

callbackcode.begin_license_wait
Begin waiting for license.

callbackcode.begin_mio
: The callback function is called when the mixed-integer optimizer is started.

callbackcode.begin_optimizer
: The callback function is called when the optimizer is started.

callbackcode.begin_presolve
: The callback function is called when the presolve is started.

callbackcode.begin_primal_bi
: The callback function is called from within the basis identification procedure when the primal phase is started.

callbackcode.begin_primal_repair
: Begin primal feasibility repair.

callbackcode.begin_primal_sensitivity
: Primal sensitivity analysis is started.

callbackcode.begin_primal_setup_bi
: The callback function is called when the primal BI setup is started.

callbackcode.begin_primal_simplex
: The callback function is called when the primal simplex optimizer is started.

callbackcode.begin_primal_simplex_bi
: The callback function is called from within the basis identification procedure when the primal simplex clean-up phase is started.

callbackcode.begin_qcqo_reformulate
: Begin QCQO reformulation.

callbackcode.begin_read
: **MOSEK** has started reading a problem file.

callbackcode.begin_root_cutgen
: The callback function is called when root cut generation is started.

callbackcode.begin_simplex
: The callback function is called when the simplex optimizer is started.

callbackcode.begin_simplex_bi
: The callback function is called from within the basis identification procedure when the simplex clean-up phase is started.

callbackcode.begin_to_conic
: Begin conic reformulation.

callbackcode.begin_write
: **MOSEK** has started writing a problem file.

callbackcode.conic
: The callback function is called from within the conic optimizer after the information database has been updated.

callbackcode.dual_simplex
: The callback function is called from within the dual simplex optimizer.

callbackcode.end_bi
: The callback function is called when the basis identification procedure is terminated.

callbackcode.end_conic
: The callback function is called when the conic optimizer is terminated.

callbackcode.end_dual_bi
: The callback function is called from within the basis identification procedure when the dual phase is terminated.

callbackcode.end_dual_sensitivity
: Dual sensitivity analysis is terminated.

callbackcode.end_dual_setup_bi
: The callback function is called when the dual BI phase is terminated.

callbackcode.end_dual_simplex
The callback function is called when the dual simplex optimizer is terminated.

callbackcode.end_dual_simplex_bi
The callback function is called from within the basis identification procedure when the dual clean-up phase is terminated.

callbackcode.end_full_convexity_check
End full convexity check.

callbackcode.end_infeas_ana
The callback function is called when the infeasibility analyzer is terminated.

callbackcode.end_intpnt
The callback function is called when the interior-point optimizer is terminated.

callbackcode.end_license_wait
End waiting for license.

callbackcode.end_mio
The callback function is called when the mixed-integer optimizer is terminated.

callbackcode.end_optimizer
The callback function is called when the optimizer is terminated.

callbackcode.end_presolve
The callback function is called when the presolve is completed.

callbackcode.end_primal_bi
The callback function is called from within the basis identification procedure when the primal phase is terminated.

callbackcode.end_primal_repair
End primal feasibility repair.

callbackcode.end_primal_sensitivity
Primal sensitivity analysis is terminated.

callbackcode.end_primal_setup_bi
The callback function is called when the primal BI setup is terminated.

callbackcode.end_primal_simplex
The callback function is called when the primal simplex optimizer is terminated.

callbackcode.end_primal_simplex_bi
The callback function is called from within the basis identification procedure when the primal clean-up phase is terminated.

callbackcode.end_qcqo_reformulate
End QCQO reformulation.

callbackcode.end_read
MOSEK has finished reading a problem file.

callbackcode.end_root_cutgen
The callback function is called when root cut generation is terminated.

callbackcode.end_simplex
The callback function is called when the simplex optimizer is terminated.

callbackcode.end_simplex_bi
The callback function is called from within the basis identification procedure when the simplex clean-up phase is terminated.

callbackcode.end_to_conic
End conic reformulation.

callbackcode.end_write
MOSEK has finished writing a problem file.

callbackcode.im_bi
The callback function is called from within the basis identification procedure at an intermediate point.

callbackcode.im_conic
: The callback function is called at an intermediate stage within the conic optimizer where the information database has not been updated.

callbackcode.im_dual_bi
: The callback function is called from within the basis identification procedure at an intermediate point in the dual phase.

callbackcode.im_dual_sensivity
: The callback function is called at an intermediate stage of the dual sensitivity analysis.

callbackcode.im_dual_simplex
: The callback function is called at an intermediate point in the dual simplex optimizer.

callbackcode.im_full_convexity_check
: The callback function is called at an intermediate stage of the full convexity check.

callbackcode.im_intpnt
: The callback function is called at an intermediate stage within the interior-point optimizer where the information database has not been updated.

callbackcode.im_license_wait
: **MOSEK** is waiting for a license.

callbackcode.im_lu
: The callback function is called from within the LU factorization procedure at an intermediate point.

callbackcode.im_mio
: The callback function is called at an intermediate point in the mixed-integer optimizer.

callbackcode.im_mio_dual_simplex
: The callback function is called at an intermediate point in the mixed-integer optimizer while running the dual simplex optimizer.

callbackcode.im_mio_intpnt
: The callback function is called at an intermediate point in the mixed-integer optimizer while running the interior-point optimizer.

callbackcode.im_mio_primal_simplex
: The callback function is called at an intermediate point in the mixed-integer optimizer while running the primal simplex optimizer.

callbackcode.im_order
: The callback function is called from within the matrix ordering procedure at an intermediate point.

callbackcode.im_presolve
: The callback function is called from within the presolve procedure at an intermediate stage.

callbackcode.im_primal_bi
: The callback function is called from within the basis identification procedure at an intermediate point in the primal phase.

callbackcode.im_primal_sensivity
: The callback function is called at an intermediate stage of the primal sensitivity analysis.

callbackcode.im_primal_simplex
: The callback function is called at an intermediate point in the primal simplex optimizer.

callbackcode.im_qo_reformulate
: The callback function is called at an intermediate stage of the conic quadratic reformulation.

callbackcode.im_read
: Intermediate stage in reading.

callbackcode.im_root_cutgen
: The callback is called from within root cut generation at an intermediate stage.

callbackcode.im_simplex
: The callback function is called from within the simplex optimizer at an intermediate point.

callbackcode.im_simplex_bi
The callback function is called from within the basis identification procedure at an intermediate point in the simplex clean-up phase. The frequency of the callbacks is controlled by the *iparam.log_sim_freq* parameter.

callbackcode.intpnt
The callback function is called from within the interior-point optimizer after the information database has been updated.

callbackcode.new_int_mio
The callback function is called after a new integer solution has been located by the mixed-integer optimizer.

callbackcode.primal_simplex
The callback function is called from within the primal simplex optimizer.

callbackcode.read_opf
The callback function is called from the OPF reader.

callbackcode.read_opf_section
A chunk of Q non-zeros has been read from a problem file.

callbackcode.solving_remote
The callback function is called while the task is being solved on a remote server.

callbackcode.update_dual_bi
The callback function is called from within the basis identification procedure at an intermediate point in the dual phase.

callbackcode.update_dual_simplex
The callback function is called in the dual simplex optimizer.

callbackcode.update_dual_simplex_bi
The callback function is called from within the basis identification procedure at an intermediate point in the dual simplex clean-up phase. The frequency of the callbacks is controlled by the *iparam.log_sim_freq* parameter.

callbackcode.update_presolve
The callback function is called from within the presolve procedure.

callbackcode.update_primal_bi
The callback function is called from within the basis identification procedure at an intermediate point in the primal phase.

callbackcode.update_primal_simplex
The callback function is called in the primal simplex optimizer.

callbackcode.update_primal_simplex_bi
The callback function is called from within the basis identification procedure at an intermediate point in the primal simplex clean-up phase. The frequency of the callbacks is controlled by the *iparam.log_sim_freq* parameter.

callbackcode.write_opf
The callback function is called from the OPF writer.

checkconvexitytype
Types of convexity checks.

checkconvexitytype.none
No convexity check.

checkconvexitytype.simple
Perform simple and fast convexity check.

checkconvexitytype.full
Perform a full convexity check.

compresstype
Compression types

compresstype.none
No compression is used.

`compresstype.free`
: The type of compression used is chosen automatically.

`compresstype.gzip`
: The type of compression used is gzip compatible.

`compresstype.zstd`
: The type of compression used is zstd compatible.

`conetype`
: Cone types

`conetype.quad`
: The cone is a quadratic cone.

`conetype.rquad`
: The cone is a rotated quadratic cone.

`conetype.pexp`
: A primal exponential cone.

`conetype.dexp`
: A dual exponential cone.

`conetype.ppow`
: A primal power cone.

`conetype.dpow`
: A dual power cone.

`conetype.zero`
: The zero cone.

`nametype`
: Name types

`nametype.gen`
: General names. However, no duplicate and blank names are allowed.

`nametype.mps`
: MPS type names.

`nametype.lp`
: LP type names.

`scopr`
: SCopt operator types

`scopr.ent`
: Entropy

`scopr.exp`
: Exponential

`scopr.log`
: Logarithm

`scopr.pow`
: Power

`scopr.sqrt`
: Square root

`symmattype`
: Cone types

`symmattype.sparse`
: Sparse symmetric matrix.

`dataformat`
: Data format types

`dataformat.extension`
: The file extension is used to determine the data file format.

`dataformat.mps`
: The data file is MPS formatted.

dataformat.lp
The data file is LP formatted.

dataformat.op
The data file is an optimization problem formatted file.

dataformat.free_mps
The data a free MPS formatted file.

dataformat.task
Generic task dump file.

dataformat.ptf
(P)retty (T)ext (F)format.

dataformat.cb
Conic benchmark format,

dataformat.json_task
JSON based task format.

dinfitem
Double information items

dinfitem.bi_clean_dual_time
Time spent within the dual clean-up optimizer of the basis identification procedure since its invocation.

dinfitem.bi_clean_primal_time
Time spent within the primal clean-up optimizer of the basis identification procedure since its invocation.

dinfitem.bi_clean_time
Time spent within the clean-up phase of the basis identification procedure since its invocation.

dinfitem.bi_dual_time
Time spent within the dual phase basis identification procedure since its invocation.

dinfitem.bi_primal_time
Time spent within the primal phase of the basis identification procedure since its invocation.

dinfitem.bi_time
Time spent within the basis identification procedure since its invocation.

dinfitem.intpnt_dual_feas
Dual feasibility measure reported by the interior-point optimizer. (For the interior-point optimizer this measure is not directly related to the original problem because a homogeneous model is employed.)

dinfitem.intpnt_dual_obj
Dual objective value reported by the interior-point optimizer.

dinfitem.intpnt_factor_num_flops
An estimate of the number of flops used in the factorization.

dinfitem.intpnt_opt_status
A measure of optimality of the solution. It should converge to $+1$ if the problem has a primal-dual optimal solution, and converge to -1 if the problem is (strictly) primal or dual infeasible. If the measure converges to another constant, or fails to settle, the problem is usually ill-posed.

dinfitem.intpnt_order_time
Order time (in seconds).

dinfitem.intpnt_primal_feas
Primal feasibility measure reported by the interior-point optimizer. (For the interior-point optimizer this measure is not directly related to the original problem because a homogeneous model is employed).

dinfitem.intpnt_primal_obj
Primal objective value reported by the interior-point optimizer.

dinfitem.intpnt_time
Time spent within the interior-point optimizer since its invocation.

dinfitem.mio_clique_separation_time
: Separation time for clique cuts.

dinfitem.mio_cmir_separation_time
: Separation time for CMIR cuts.

dinfitem.mio_construct_solution_obj
: If **MOSEK** has successfully constructed an integer feasible solution, then this item contains the optimal objective value corresponding to the feasible solution.

dinfitem.mio_dual_bound_after_presolve
: Value of the dual bound after presolve but before cut generation.

dinfitem.mio_gmi_separation_time
: Separation time for GMI cuts.

dinfitem.mio_implied_bound_time
: Separation time for implied bound cuts.

dinfitem.mio_knapsack_cover_separation_time
: Separation time for knapsack cover.

dinfitem.mio_obj_abs_gap
: Given the mixed-integer optimizer has computed a feasible solution and a bound on the optimal objective value, then this item contains the absolute gap defined by

$$|(\text{objective value of feasible solution}) - (\text{objective bound})|.$$

Otherwise it has the value -1.0.

dinfitem.mio_obj_bound
: The best known bound on the objective function. This value is undefined until at least one relaxation has been solved: To see if this is the case check that *iinfitem.mio_num_relax* is strictly positive.

dinfitem.mio_obj_int
: The primal objective value corresponding to the best integer feasible solution. Please note that at least one integer feasible solution must have been located i.e. check *iinfitem.mio_num_int_solutions*.

dinfitem.mio_obj_rel_gap
: Given that the mixed-integer optimizer has computed a feasible solution and a bound on the optimal objective value, then this item contains the relative gap defined by

$$\frac{|(\text{objective value of feasible solution}) - (\text{objective bound})|}{\max(\delta, |(\text{objective value of feasible solution})|)}.$$

where δ is given by the parameter *dparam.mio_rel_gap_const*. Otherwise it has the value -1.0.

dinfitem.mio_probing_time
: Total time for probing.

dinfitem.mio_root_cutgen_time
: Total time for cut generation.

dinfitem.mio_root_optimizer_time
: Time spent in the optimizer while solving the root node relaxation

dinfitem.mio_root_presolve_time
: Time spent presolving the problem at the root node.

dinfitem.mio_time
: Time spent in the mixed-integer optimizer.

dinfitem.mio_user_obj_cut
: If the objective cut is used, then this information item has the value of the cut.

dinfitem.optimizer_time
: Total time spent in the optimizer since it was invoked.

`dinfitem.presolve_eli_time`
Total time spent in the eliminator since the presolve was invoked.

`dinfitem.presolve_lindep_time`
Total time spent in the linear dependency checker since the presolve was invoked.

`dinfitem.presolve_time`
Total time (in seconds) spent in the presolve since it was invoked.

`dinfitem.primal_repair_penalty_obj`
The optimal objective value of the penalty function.

`dinfitem.qcqo_reformulate_max_perturbation`
Maximum absolute diagonal perturbation occurring during the QCQO reformulation.

`dinfitem.qcqo_reformulate_time`
Time spent with conic quadratic reformulation.

`dinfitem.qcqo_reformulate_worst_cholesky_column_scaling`
Worst Cholesky column scaling.

`dinfitem.qcqo_reformulate_worst_cholesky_diag_scaling`
Worst Cholesky diagonal scaling.

`dinfitem.rd_time`
Time spent reading the data file.

`dinfitem.sim_dual_time`
Time spent in the dual simplex optimizer since invoking it.

`dinfitem.sim_feas`
Feasibility measure reported by the simplex optimizer.

`dinfitem.sim_obj`
Objective value reported by the simplex optimizer.

`dinfitem.sim_primal_time`
Time spent in the primal simplex optimizer since invoking it.

`dinfitem.sim_time`
Time spent in the simplex optimizer since invoking it.

`dinfitem.sol_bas_dual_obj`
Dual objective value of the basic solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.updatesolutioninfo*.

`dinfitem.sol_bas_dviolcon`
Maximal dual bound violation for x^c in the basic solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.updatesolutioninfo*.

`dinfitem.sol_bas_dviolvar`
Maximal dual bound violation for x^x in the basic solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.updatesolutioninfo*.

`dinfitem.sol_bas_nrm_barx`
Infinity norm of $\overline{X}$ in the basic solution.

`dinfitem.sol_bas_nrm_slc`
Infinity norm of s_l^c in the basic solution.

`dinfitem.sol_bas_nrm_slx`
Infinity norm of s_l^x in the basic solution.

`dinfitem.sol_bas_nrm_suc`
Infinity norm of s_u^c in the basic solution.

`dinfitem.sol_bas_nrm_sux`
Infinity norm of s_u^X in the basic solution.

`dinfitem.sol_bas_nrm_xc`
Infinity norm of x^c in the basic solution.

`dinfitem.sol_bas_nrm_xx`
Infinity norm of x^x in the basic solution.

dinfitem.sol_bas_nrm_y
: Infinity norm of y in the basic solution.

dinfitem.sol_bas_primal_obj
: Primal objective value of the basic solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.updatesolutioninfo*.

dinfitem.sol_bas_pviolcon
: Maximal primal bound violation for x^c in the basic solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.updatesolutioninfo*.

dinfitem.sol_bas_pviolvar
: Maximal primal bound violation for x^x in the basic solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.updatesolutioninfo*.

dinfitem.sol_itg_nrm_barx
: Infinity norm of $\overline{X}$ in the integer solution.

dinfitem.sol_itg_nrm_xc
: Infinity norm of x^c in the integer solution.

dinfitem.sol_itg_nrm_xx
: Infinity norm of x^x in the integer solution.

dinfitem.sol_itg_primal_obj
: Primal objective value of the integer solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.updatesolutioninfo*.

dinfitem.sol_itg_pviolbarvar
: Maximal primal bound violation for $\overline{X}$ in the integer solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.updatesolutioninfo*.

dinfitem.sol_itg_pviolcon
: Maximal primal bound violation for x^c in the integer solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.updatesolutioninfo*.

dinfitem.sol_itg_pviolcones
: Maximal primal violation for primal conic constraints in the integer solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.updatesolutioninfo*.

dinfitem.sol_itg_pviolitg
: Maximal violation for the integer constraints in the integer solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.updatesolutioninfo*.

dinfitem.sol_itg_pviolvar
: Maximal primal bound violation for x^x in the integer solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.updatesolutioninfo*.

dinfitem.sol_itr_dual_obj
: Dual objective value of the interior-point solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.updatesolutioninfo*.

dinfitem.sol_itr_dviolbarvar
: Maximal dual bound violation for $\overline{X}$ in the interior-point solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.updatesolutioninfo*.

dinfitem.sol_itr_dviolcon
: Maximal dual bound violation for x^c in the interior-point solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.updatesolutioninfo*.

dinfitem.sol_itr_dviolcones
: Maximal dual violation for dual conic constraints in the interior-point solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.updatesolutioninfo*.

dinfitem.sol_itr_dviolvar
: Maximal dual bound violation for x^x in the interior-point solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.updatesolutioninfo*.

dinfitem.sol_itr_nrm_bars
: Infinity norm of $\overline{S}$ in the interior-point solution.

`dinfitem.sol_itr_nrm_barx`
: Infinity norm of $\overline{X}$ in the interior-point solution.

`dinfitem.sol_itr_nrm_slc`
: Infinity norm of s_l^c in the interior-point solution.

`dinfitem.sol_itr_nrm_slx`
: Infinity norm of s_l^x in the interior-point solution.

`dinfitem.sol_itr_nrm_snx`
: Infinity norm of s_n^x in the interior-point solution.

`dinfitem.sol_itr_nrm_suc`
: Infinity norm of s_u^c in the interior-point solution.

`dinfitem.sol_itr_nrm_sux`
: Infinity norm of s_u^X in the interior-point solution.

`dinfitem.sol_itr_nrm_xc`
: Infinity norm of x^c in the interior-point solution.

`dinfitem.sol_itr_nrm_xx`
: Infinity norm of x^x in the interior-point solution.

`dinfitem.sol_itr_nrm_y`
: Infinity norm of y in the interior-point solution.

`dinfitem.sol_itr_primal_obj`
: Primal objective value of the interior-point solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.updatesolutioninfo* .

`dinfitem.sol_itr_pviolbarvar`
: Maximal primal bound violation for $\overline{X}$ in the interior-point solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.updatesolutioninfo* .

`dinfitem.sol_itr_pviolcon`
: Maximal primal bound violation for x^c in the interior-point solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.updatesolutioninfo* .

`dinfitem.sol_itr_pviolcones`
: Maximal primal violation for primal conic constraints in the interior-point solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.updatesolutioninfo* .

`dinfitem.sol_itr_pviolvar`
: Maximal primal bound violation for x^x in the interior-point solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.updatesolutioninfo* .

`dinfitem.to_conic_time`
: Time spent in the last to conic reformulation.

`feature`
: License feature

`feature.pts`
: Base system.

`feature.pton`
: Conic extension.

`liinfitem`
: Long integer information items.

`liinfitem.bi_clean_dual_deg_iter`
: Number of dual degenerate clean iterations performed in the basis identification.

`liinfitem.bi_clean_dual_iter`
: Number of dual clean iterations performed in the basis identification.

`liinfitem.bi_clean_primal_deg_iter`
: Number of primal degenerate clean iterations performed in the basis identification.

`liinfitem.bi_clean_primal_iter`
: Number of primal clean iterations performed in the basis identification.

liinfitem.bi_dual_iter
Number of dual pivots performed in the basis identification.

liinfitem.bi_primal_iter
Number of primal pivots performed in the basis identification.

liinfitem.intpnt_factor_num_nz
Number of non-zeros in factorization.

liinfitem.mio_anz
Number of non-zero entries in the constraint matrix of the probelm to be solved by the mixed-integer optimizer.

liinfitem.mio_intpnt_iter
Number of interior-point iterations performed by the mixed-integer optimizer.

liinfitem.mio_presolved_anz
Number of non-zero entries in the constraint matrix of the problem after the mixed-integer optimizer's presolve.

liinfitem.mio_simplex_iter
Number of simplex iterations performed by the mixed-integer optimizer.

liinfitem.rd_numanz
Number of non-zeros in A that is read.

liinfitem.rd_numqnz
Number of Q non-zeros.

iinfitem
Integer information items.

iinfitem.ana_pro_num_con
Number of constraints in the problem. This value is set by *Task.analyzeproblem*.

iinfitem.ana_pro_num_con_eq
Number of equality constraints. This value is set by *Task.analyzeproblem*.

iinfitem.ana_pro_num_con_fr
Number of unbounded constraints. This value is set by *Task.analyzeproblem*.

iinfitem.ana_pro_num_con_lo
Number of constraints with a lower bound and an infinite upper bound. This value is set by *Task.analyzeproblem*.

iinfitem.ana_pro_num_con_ra
Number of constraints with finite lower and upper bounds. This value is set by *Task.analyzeproblem*.

iinfitem.ana_pro_num_con_up
Number of constraints with an upper bound and an infinite lower bound. This value is set by *Task.analyzeproblem*.

iinfitem.ana_pro_num_var
Number of variables in the problem. This value is set by *Task.analyzeproblem*.

iinfitem.ana_pro_num_var_bin
Number of binary (0-1) variables. This value is set by *Task.analyzeproblem*.

iinfitem.ana_pro_num_var_cont
Number of continuous variables. This value is set by *Task.analyzeproblem*.

iinfitem.ana_pro_num_var_eq
Number of fixed variables. This value is set by *Task.analyzeproblem*.

iinfitem.ana_pro_num_var_fr
Number of free variables. This value is set by *Task.analyzeproblem*.

iinfitem.ana_pro_num_var_int
Number of general integer variables. This value is set by *Task.analyzeproblem*.

iinfitem.ana_pro_num_var_lo
Number of variables with a lower bound and an infinite upper bound. This value is set by *Task.analyzeproblem*.

iinfitem.ana_pro_num_var_ra
: Number of variables with finite lower and upper bounds. This value is set by *Task.analyzeproblem*.

iinfitem.ana_pro_num_var_up
: Number of variables with an upper bound and an infinite lower bound. This value is set by *Task.analyzeproblem*.

iinfitem.intpnt_factor_dim_dense
: Dimension of the dense sub system in factorization.

iinfitem.intpnt_iter
: Number of interior-point iterations since invoking the interior-point optimizer.

iinfitem.intpnt_num_threads
: Number of threads that the interior-point optimizer is using.

iinfitem.intpnt_solve_dual
: Non-zero if the interior-point optimizer is solving the dual problem.

iinfitem.mio_absgap_satisfied
: Non-zero if absolute gap is within tolerances.

iinfitem.mio_clique_table_size
: Size of the clique table.

iinfitem.mio_construct_solution
: This item informs if **MOSEK** constructed an initial integer feasible solution.

  - -1: tried, but failed,
  - 0: no partial solution supplied by the user,
  - 1: constructed feasible solution.

iinfitem.mio_node_depth
: Depth of the last node solved.

iinfitem.mio_num_active_nodes
: Number of active branch and bound nodes.

iinfitem.mio_num_branch
: Number of branches performed during the optimization.

iinfitem.mio_num_clique_cuts
: Number of clique cuts.

iinfitem.mio_num_cmir_cuts
: Number of Complemented Mixed Integer Rounding (CMIR) cuts.

iinfitem.mio_num_gomory_cuts
: Number of Gomory cuts.

iinfitem.mio_num_implied_bound_cuts
: Number of implied bound cuts.

iinfitem.mio_num_int_solutions
: Number of integer feasible solutions that have been found.

iinfitem.mio_num_knapsack_cover_cuts
: Number of clique cuts.

iinfitem.mio_num_relax
: Number of relaxations solved during the optimization.

iinfitem.mio_num_repeated_presolve
: Number of times presolve was repeated at root.

iinfitem.mio_numbin
: Number of binary variables in the problem to be solved by the mixed-integer optimizer.

iinfitem.mio_numbinconevar
: Number of binary cone variables in the problem to be solved by the mixed-integer optimizer.

iinfitem.mio_numcon
: Number of constraints in the problem to be solved by the mixed-integer optimizer.

iinfitem.mio_numcone
Number of cones in the problem to be solved by the mixed-integer optimizer.

iinfitem.mio_numconevar
Number of cone variables in the problem to be solved by the mixed-integer optimizer.

iinfitem.mio_numcont
Number of continuous variables in the problem to be solved by the mixed-integer optimizer.

iinfitem.mio_numcontconevar
Number of continuous cone variables in the problem to be solved by the mixed-integer optimizer.

iinfitem.mio_numdexpcones
Number of dual exponential cones in the problem to be solved by the mixed-integer optimizer.

iinfitem.mio_numdpowcones
Number of dual power cones in the problem to be solved by the mixed-integer optimizer.

iinfitem.mio_numint
Number of integer variables in the problem to be solved by the mixed-integer optimizer.

iinfitem.mio_numintconevar
Number of integer cone variables in the problem to be solved by the mixed-integer optimizer.

iinfitem.mio_numpexpcones
Number of primal exponential cones in the problem to be solved by the mixed-integer optimizer.

iinfitem.mio_numppowcones
Number of primal power cones in the problem to be solved by the mixed-integer optimizer.

iinfitem.mio_numqcones
Number of quadratic cones in the problem to be solved by the mixed-integer optimizer.

iinfitem.mio_numrqcones
Number of rotated quadratic cones in the problem to be solved by the mixed-integer optimizer.

iinfitem.mio_numvar
Number of variables in the problem to be solved by the mixed-integer optimizer.

iinfitem.mio_obj_bound_defined
Non-zero if a valid objective bound has been found, otherwise zero.

iinfitem.mio_presolved_numbin
Number of binary variables in the problem after the mixed-integer optimizer's presolve.

iinfitem.mio_presolved_numbinconevar
Number of binary cone variables in the problem after the mixed-integer optimizer's presolve.

iinfitem.mio_presolved_numcon
Number of constraints in the problem after the mixed-integer optimizer's presolve.

iinfitem.mio_presolved_numcone
Number of cones in the problem after the mixed-integer optimizer's presolve.

iinfitem.mio_presolved_numconevar
Number of cone variables in the problem after the mixed-integer optimizer's presolve.

iinfitem.mio_presolved_numcont
Number of continuous variables in the problem after the mixed-integer optimizer's presolve.

iinfitem.mio_presolved_numcontconevar
Number of continuous cone variables in the problem after the mixed-integer optimizer's presolve.

iinfitem.mio_presolved_numdexpcones
Number of dual exponential cones in the problem after the mixed-integer optimizer's presolve.

iinfitem.mio_presolved_numdpowcones
Number of dual power cones in the problem after the mixed-integer optimizer's presolve.

iinfitem.mio_presolved_numint
Number of integer variables in the problem after the mixed-integer optimizer's presolve.

iinfitem.mio_presolved_numintconevar
: Number of integer cone variables in the problem after the mixed-integer optimizer's presolve.

iinfitem.mio_presolved_numpexpcones
: Number of primal exponential cones in the problem after the mixed-integer optimizer's presolve.

iinfitem.mio_presolved_numppowcones
: Number of primal power cones in the problem after the mixed-integer optimizer's presolve.

iinfitem.mio_presolved_numqcones
: Number of quadratic cones in the problem after the mixed-integer optimizer's presolve.

iinfitem.mio_presolved_numrqcones
: Number of rotated quadratic cones in the problem after the mixed-integer optimizer's presolve.

iinfitem.mio_presolved_numvar
: Number of variables in the problem after the mixed-integer optimizer's presolve.

iinfitem.mio_relgap_satisfied
: Non-zero if relative gap is within tolerances.

iinfitem.mio_total_num_cuts
: Total number of cuts generated by the mixed-integer optimizer.

iinfitem.mio_user_obj_cut
: If it is non-zero, then the objective cut is used.

iinfitem.opt_numcon
: Number of constraints in the problem solved when the optimizer is called.

iinfitem.opt_numvar
: Number of variables in the problem solved when the optimizer is called

iinfitem.optimize_response
: The response code returned by optimize.

iinfitem.purify_dual_success
: Is nonzero if the dual solution is purified.

iinfitem.purify_primal_success
: Is nonzero if the primal solution is purified.

iinfitem.rd_numbarvar
: Number of symmetric variables read.

iinfitem.rd_numcon
: Number of constraints read.

iinfitem.rd_numcone
: Number of conic constraints read.

iinfitem.rd_numintvar
: Number of integer-constrained variables read.

iinfitem.rd_numq
: Number of nonempty Q matrices read.

iinfitem.rd_numvar
: Number of variables read.

iinfitem.rd_protype
: Problem type.

iinfitem.sim_dual_deg_iter
: The number of dual degenerate iterations.

iinfitem.sim_dual_hotstart
: If 1 then the dual simplex algorithm is solving from an advanced basis.

iinfitem.sim_dual_hotstart_lu
: If 1 then a valid basis factorization of full rank was located and used by the dual simplex algorithm.

iinfitem.sim_dual_inf_iter
: The number of iterations taken with dual infeasibility.

iinfitem.sim_dual_iter
: Number of dual simplex iterations during the last optimization.

iinfitem.sim_numcon
: Number of constraints in the problem solved by the simplex optimizer.

iinfitem.sim_numvar
: Number of variables in the problem solved by the simplex optimizer.

iinfitem.sim_primal_deg_iter
: The number of primal degenerate iterations.

iinfitem.sim_primal_hotstart
: If 1 then the primal simplex algorithm is solving from an advanced basis.

iinfitem.sim_primal_hotstart_lu
: If 1 then a valid basis factorization of full rank was located and used by the primal simplex algorithm.

iinfitem.sim_primal_inf_iter
: The number of iterations taken with primal infeasibility.

iinfitem.sim_primal_iter
: Number of primal simplex iterations during the last optimization.

iinfitem.sim_solve_dual
: Is non-zero if dual problem is solved.

iinfitem.sol_bas_prosta
: Problem status of the basic solution. Updated after each optimization.

iinfitem.sol_bas_solsta
: Solution status of the basic solution. Updated after each optimization.

iinfitem.sol_itg_prosta
: Problem status of the integer solution. Updated after each optimization.

iinfitem.sol_itg_solsta
: Solution status of the integer solution. Updated after each optimization.

iinfitem.sol_itr_prosta
: Problem status of the interior-point solution. Updated after each optimization.

iinfitem.sol_itr_solsta
: Solution status of the interior-point solution. Updated after each optimization.

iinfitem.sto_num_a_realloc
: Number of times the storage for storing A has been changed. A large value may indicates that memory fragmentation may occur.

inftype
: Information item types

inftype.dou_type
: Is a double information type.

inftype.int_type
: Is an integer.

inftype.lint_type
: Is a long integer.

iomode
: Input/output modes

iomode.read
: The file is read-only.

iomode.write
: The file is write-only. If the file exists then it is truncated when it is opened. Otherwise it is created when it is opened.

`iomode.readwrite`
: The file is to read and write.

`branchdir`
: Specifies the branching direction.

`branchdir.free`
: The mixed-integer optimizer decides which branch to choose.

`branchdir.up`
: The mixed-integer optimizer always chooses the up branch first.

`branchdir.down`
: The mixed-integer optimizer always chooses the down branch first.

`branchdir.near`
: Branch in direction nearest to selected fractional variable.

`branchdir.far`
: Branch in direction farthest from selected fractional variable.

`branchdir.root_lp`
: Chose direction based on root lp value of selected variable.

`branchdir.guided`
: Branch in direction of current incumbent.

`branchdir.pseudocost`
: Branch based on the pseudocost of the variable.

`miocontsoltype`
: Continuous mixed-integer solution type

`miocontsoltype.none`
: No interior-point or basic solution are reported when the mixed-integer optimizer is used.

`miocontsoltype.root`
: The reported interior-point and basic solutions are a solution to the root node problem when mixed-integer optimizer is used.

`miocontsoltype.itg`
: The reported interior-point and basic solutions are a solution to the problem with all integer variables fixed at the value they have in the integer solution. A solution is only reported in case the problem has a primal feasible solution.

`miocontsoltype.itg_rel`
: In case the problem is primal feasible then the reported interior-point and basic solutions are a solution to the problem with all integer variables fixed at the value they have in the integer solution. If the problem is primal infeasible, then the solution to the root node problem is reported.

`miomode`
: Integer restrictions

`miomode.ignored`
: The integer constraints are ignored and the problem is solved as a continuous problem.

`miomode.satisfied`
: Integer restrictions should be satisfied.

`mionodeseltype`
: Mixed-integer node selection types

`mionodeseltype.free`
: The optimizer decides the node selection strategy.

`mionodeseltype.first`
: The optimizer employs a depth first node selection strategy.

`mionodeseltype.best`
: The optimizer employs a best bound node selection strategy.

`mionodeseltype.pseudo`
: The optimizer employs selects the node based on a pseudo cost estimate.

`mpsformat`
MPS file format type

`mpsformat.strict`
It is assumed that the input file satisfies the MPS format strictly.

`mpsformat.relaxed`
It is assumed that the input file satisfies a slightly relaxed version of the MPS format.

`mpsformat.free`
It is assumed that the input file satisfies the free MPS format. This implies that spaces are not allowed in names. Otherwise the format is free.

`mpsformat.cplex`
The CPLEX compatible version of the MPS format is employed.

`objsense`
Objective sense types

`objsense.minimize`
The problem should be minimized.

`objsense.maximize`
The problem should be maximized.

`onoffkey`
On/off

`onoffkey.on`
Switch the option on.

`onoffkey.off`
Switch the option off.

`optimizertype`
Optimizer types

`optimizertype.conic`
The optimizer for problems having conic constraints.

`optimizertype.dual_simplex`
The dual simplex optimizer is used.

`optimizertype.free`
The optimizer is chosen automatically.

`optimizertype.free_simplex`
One of the simplex optimizers is used.

`optimizertype.intpnt`
The interior-point optimizer is used.

`optimizertype.mixed_int`
The mixed-integer optimizer.

`optimizertype.primal_simplex`
The primal simplex optimizer is used.

`orderingtype`
Ordering strategies

`orderingtype.free`
The ordering method is chosen automatically.

`orderingtype.appminloc`
Approximate minimum local fill-in ordering is employed.

`orderingtype.experimental`
This option should not be used.

`orderingtype.try_graphpar`
Always try the graph partitioning based ordering.

`orderingtype.force_graphpar`
Always use the graph partitioning based ordering even if it is worse than the approximate minimum local fill ordering.

orderingtype.none

No ordering is used.

presolvemode

Presolve method.

presolvemode.off

The problem is not presolved before it is optimized.

presolvemode.on

The problem is presolved before it is optimized.

presolvemode.free

It is decided automatically whether to presolve before the problem is optimized.

parametertype

Parameter type

parametertype.invalid_type

Not a valid parameter.

parametertype.dou_type

Is a double parameter.

parametertype.int_type

Is an integer parameter.

parametertype.str_type

Is a string parameter.

problemitem

Problem data items

problemitem.var

Item is a variable.

problemitem.con

Item is a constraint.

problemitem.cone

Item is a cone.

problemtype

Problem types

problemtype.lo

The problem is a linear optimization problem.

problemtype.qo

The problem is a quadratic optimization problem.

problemtype.qcqo

The problem is a quadratically constrained optimization problem.

problemtype.conic

A conic optimization.

problemtype.mixed

General nonlinear constraints and conic constraints. This combination can not be solved by **MOSEK**.

prosta

Problem status keys

prosta.unknown

Unknown problem status.

prosta.prim_and_dual_feas

The problem is primal and dual feasible.

prosta.prim_feas

The problem is primal feasible.

prosta.dual_feas

The problem is dual feasible.

`prosta.prim_infeas`
: The problem is primal infeasible.

`prosta.dual_infeas`
: The problem is dual infeasible.

`prosta.prim_and_dual_infeas`
: The problem is primal and dual infeasible.

`prosta.ill_posed`
: The problem is ill-posed. For example, it may be primal and dual feasible but have a positive duality gap.

`prosta.prim_infeas_or_unbounded`
: The problem is either primal infeasible or unbounded. This may occur for mixed-integer problems.

`xmlwriteroutputtype`
: XML writer output mode

`xmlwriteroutputtype.row`
: Write in row order.

`xmlwriteroutputtype.col`
: Write in column order.

`rescodetype`
: Response code type

`rescodetype.ok`
: The response code is OK.

`rescodetype.wrn`
: The response code is a warning.

`rescodetype.trm`
: The response code is an optimizer termination status.

`rescodetype.err`
: The response code is an error.

`rescodetype.unk`
: The response code does not belong to any class.

`scalingtype`
: Scaling type

`scalingtype.free`
: The optimizer chooses the scaling heuristic.

`scalingtype.none`
: No scaling is performed.

`scalingtype.moderate`
: A conservative scaling is performed.

`scalingtype.aggressive`
: A very aggressive scaling is performed.

`scalingmethod`
: Scaling method

`scalingmethod.pow2`
: Scales only with power of 2 leaving the mantissa untouched.

`scalingmethod.free`
: The optimizer chooses the scaling heuristic.

`sensitivitytype`
: Sensitivity types

`sensitivitytype.basis`
: Basis sensitivity analysis is performed.

`simseltype`
: Simplex selection strategy

simseltype.free
: The optimizer chooses the pricing strategy.

simseltype.full
: The optimizer uses full pricing.

simseltype.ase
: The optimizer uses approximate steepest-edge pricing.

simseltype.devex
: The optimizer uses devex steepest-edge pricing (or if it is not available an approximate steep-edge selection).

simseltype.se
: The optimizer uses steepest-edge selection (or if it is not available an approximate steep-edge selection).

simseltype.partial
: The optimizer uses a partial selection approach. The approach is usually beneficial if the number of variables is much larger than the number of constraints.

solitem
: Solution items

solitem.xc
: Solution for the constraints.

solitem.xx
: Variable solution.

solitem.y
: Lagrange multipliers for equations.

solitem.slc
: Lagrange multipliers for lower bounds on the constraints.

solitem.suc
: Lagrange multipliers for upper bounds on the constraints.

solitem.slx
: Lagrange multipliers for lower bounds on the variables.

solitem.sux
: Lagrange multipliers for upper bounds on the variables.

solitem.snx
: Lagrange multipliers corresponding to the conic constraints on the variables.

solsta
: Solution status keys

solsta.unknown
: Status of the solution is unknown.

solsta.optimal
: The solution is optimal.

solsta.prim_feas
: The solution is primal feasible.

solsta.dual_feas
: The solution is dual feasible.

solsta.prim_and_dual_feas
: The solution is both primal and dual feasible.

solsta.prim_infeas_cer
: The solution is a certificate of primal infeasibility.

solsta.dual_infeas_cer
: The solution is a certificate of dual infeasibility.

solsta.prim_illposed_cer
: The solution is a certificate that the primal problem is illposed.

solsta.dual_illposed_cer
: The solution is a certificate that the dual problem is illposed.

solsta.integer_optimal
: The primal solution is integer optimal.

soltype
: Solution types

  soltype.bas
  : The basic solution.

  soltype.itr
  : The interior solution.

  soltype.itg
  : The integer solution.

solveform
: Solve primal or dual form

  solveform.free
  : The optimizer is free to solve either the primal or the dual problem.

  solveform.primal
  : The optimizer should solve the primal problem.

  solveform.dual
  : The optimizer should solve the dual problem.

stakey
: Status keys

  stakey.unk
  : The status for the constraint or variable is unknown.

  stakey.bas
  : The constraint or variable is in the basis.

  stakey.supbas
  : The constraint or variable is super basic.

  stakey.low
  : The constraint or variable is at its lower bound.

  stakey.upr
  : The constraint or variable is at its upper bound.

  stakey.fix
  : The constraint or variable is fixed.

  stakey.inf
  : The constraint or variable is infeasible in the bounds.

startpointtype
: Starting point types

  startpointtype.free
  : The starting point is chosen automatically.

  startpointtype.guess
  : The optimizer guesses a starting point.

  startpointtype.constant
  : The optimizer constructs a starting point by assigning a constant value to all primal and dual variables. This starting point is normally robust.

  startpointtype.satisfy_bounds
  : The starting point is chosen to satisfy all the simple bounds on nonlinear variables. If this starting point is employed, then more care than usual should employed when choosing the bounds on the nonlinear variables. In particular very tight bounds should be avoided.

streamtype
: Stream types

streamtype.log
: Log stream. Contains the aggregated contents of all other streams. This means that a message written to any other stream will also be written to this stream.

streamtype.msg
: Message stream. Log information relating to performance and progress of the optimization is written to this stream.

streamtype.err
: Error stream. Error messages are written to this stream.

streamtype.wrn
: Warning stream. Warning messages are written to this stream.

value
: Integer values

value.max_str_len
: Maximum string length allowed in **MOSEK**.

value.license_buffer_length
: The length of a license key buffer.

variabletype
: Variable types

variabletype.type_cont
: Is a continuous variable.

variabletype.type_int
: Is an integer variable.

15.10 Function Types

callbackfunc

```
def callbackfunc (code, dinf, iinf, liinf) -> stop
```

The progress and information callback function is a user-defined function which will be called by **MOSEK** occasionally during the optimization process. In particular, the callback function is called at the beginning of each iteration in the interior-point optimizer. For the simplex optimizers *iparam.log_sim_freq* controls how frequently the callback is called.

The user *must not* call any **MOSEK** function directly or indirectly from the callback function. The only exception is the possibility to retrieve an integer solution, see *Progress and data callback*.

Parameters

- code (*callbackcode*) – Callback code indicating current operation of the solver. (input)
- dinf (float[]) – Array of double information items. (input)
- iinf (int[]) – Array of integer information items. (input)
- liinf (int[]) – Array of long integer information items. (input)

Return stop (int) – Non-zero if the optimizer should be terminated; zero otherwise.

progresscallbackfunc

```
def progresscallbackfunc (code) -> stop
```

The progress callback function is a user-defined function which will be called by **MOSEK** occasionally during the optimization process. In particular, the callback function is called at the beginning of each iteration in the interior-point optimizer. For the simplex optimizers *iparam.log_sim_freq* controls how frequently the callback is called.

The user *must not* call any **MOSEK** function directly or indirectly from the callback function. If the progress callback function returns a non-zero value, the optimization process is terminated.

Parameters `code` (*mosek.callbackcode*) – Callback code indicating the current status of the solver. (input)

Return `stop` (`int`) – Non-zero if the optimizer should be terminated; zero otherwise.

`streamfunc`

```
def streamfunc (msg)
```

The message-stream callback function is a user-defined function which can be linked to any of the **MOSEK** streams. Doing so, the function is called whenever **MOSEK** sends a message to the stream.

The user *must not* call any **MOSEK** function directly or indirectly from the callback function.

Parameters `msg` (`str`) – A string containing the message. (input)

15.11 Nonlinear interfaces (obsolete)

Important: This is a legacy document for users familiar with SCopt, DGopt, EXPopt, mskenopt, mskscopt and mskgpopt from previous versions of **MOSEK**. These interfaces have now been removed. We assume familiarity with documentation included in version 8. All problems expressible with this interface can (and should) be reformulated using the exponential cone and power cones.

New users should formulate problems involving powers, logarithms and exponentials directly in conic form.

Conversion tutorial

We recommend converting all nonlinear problems using SCopt, DGopt, EXPopt, mskenopt, mskscopt and mskgpopt into conic form. Depending on the values of f, g, h either the epigraph or hypograph of a SCopt function if convex, and a bounding variable can be introduced following the basic rules below. We assume all variables are within safe bounds where the SCopt operators are defined and convex. We also assume $f > 0$.

A more comprehensive modeling guide for these types of problems can be found in the **MOSEK** Modeling Cookbook.

Powers

Consider $f(x+h)^g$. This can be reformulated using the power cone.

- If $g > 1$ then $t \geq f(x+h)^g$ is equivalent to $(t/f)^{1/g} \geq |x+h|$, that is $(t/f, 1, x+h) \in \mathcal{P}_3^{1/g, 1-1/g}$.
- If $0 < g < 1$ then $|t| \leq f(x+h)^g$ is equivalent to $(x+h, 1, t/f) \in \mathcal{P}_3^{g, 1-g}$.
- If $g < 0$ then $t \geq f(x+h)^g$ is equivalent to $(t/f)(x+h)^{-g} \geq 1$, that is $(t/f, x+h, 1) \in \mathcal{P}_3^{1/(1-g), -g/(1-g)}$.

Logarithm

The bound $t \leq f \log(gx+h)$ is equivalent to $(gx+h, 1, t/f) \in K_{\exp}$.

Entropy

The bound $t \geq fx \log x$ is equivalent to $(1, x, -t/f) \in K_{\exp}$.

Exponential

The bound $t \geq f \exp(gx+h)$ is equivalent to $(t/f, 1, gx+h) \in K_{\exp}$.

Exponential optimization (EXPopt), Geometric programming (mskgpopt)

For a basic tutorial in geometric programming (GP) see Sec. 6.8.

An exponential optimization problem in standard form consists of constraints of the type:

$$t \geq \log \left(\sum_i \exp(a_i^T x + b_i) \right).$$

This log-sum-exp bound is equivalent to

$$\sum_i \exp(a_i^T x + b_i - t) \leq 1$$

and requires bounding each exponential function as explained above.

Dual geometric optimization (DGopt)

The objective function of a dual geometric problem involves maximizing expressions of the form

$$x \log \frac{c}{x} \quad \text{and} \quad x_i \log \frac{e^T x}{x_i},$$

which can be achieved using bounds $t \leq x \log \frac{y}{x}$, that is $(t, x, y) \in K_{\exp}$.

Chapter 16

Supported File Formats

MOSEK supports a range of problem and solution formats listed in Table 16.1 and Table 16.2. The **Task format** is **MOSEK**'s native binary format and it supports all features that **MOSEK** supports. The **OPF format** is **MOSEK**'s human-readable alternative that supports nearly all features (everything except semidefinite problems). In general, text formats are significantly slower to read, but can be examined and edited directly in any text editor.

Problem formats

Table 16.1: List of supported file formats for optimization problems. The column *Conic* refers to conic problems involving the quadratic, rotated quadratic, power or exponential cone. The last two columns indicate if the format supports solutions and optimizer parameters.

Format Type	Ext.	Binary/Text	LP	QO	Conic	SDP	Sol	Param
LP	lp	plain text	X	X				
MPS	mps	plain text	X	X	X			
OPF	opf	plain text	X	X	X		X	X
PTF	ptf	plain text	X	X	X	X	X	
CBF	cbf	plain text	X		X	X		
Task format	task	binary	X	X	X	X	X	X
Jtask format	jtask	text	X	X	X	X	X	X

Solution formats

Table 16.2: List of supported solution formats.

Format Type	Ext.	Binary/Text	Description
SOL	sol	plain text	Interior Solution
	bas	plain text	Basic Solution
	int	plain text	Integer
Jsol format	jsol	text	Solution

Compression

MOSEK supports GZIP and Zstandard compression. Problem files with extension `.gz` (for GZIP) and `.zst` (for Zstandard) are assumed to be compressed when read, and are automatically compressed when written. For example, a file called

```
problem.mps.gz
```

will be considered as a GZIP compressed MPS file.

16.1 The LP File Format

MOSEK supports the LP file format with some extensions. The LP format is not a completely well-defined standard and hence different optimization packages may interpret the same LP file in slightly different ways. **MOSEK** tries to emulate as closely as possible CPLEX's behavior, but tries to stay backward compatible.

The LP file format can specify problems of the form

$$\begin{array}{lccccc} \text{minimize/maximize} & & & c^T x + \frac{1}{2} q^o(x) & & \\ \text{subject to} & l^c & \leq & Ax + \frac{1}{2} q(x) & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x, \\ & & & x_{\mathcal{J}} \text{ integer}, & & \end{array}$$

where

- $x \in \mathbb{R}^n$ is the vector of decision variables.
- $c \in \mathbb{R}^n$ is the linear term in the objective.
- $q^o :\in \mathbb{R}^n \rightarrow \mathbb{R}$ is the quadratic term in the objective where

$$q^o(x) = x^T Q^o x$$

and it is assumed that

$$Q^o = (Q^o)^T.$$

- $A \in \mathbb{R}^{m \times n}$ is the constraint matrix.
- $l^c \in \mathbb{R}^m$ is the lower limit on the activity for the constraints.
- $u^c \in \mathbb{R}^m$ is the upper limit on the activity for the constraints.
- $l^x \in \mathbb{R}^n$ is the lower limit on the activity for the variables.
- $u^x \in \mathbb{R}^n$ is the upper limit on the activity for the variables.
- $q : \mathbb{R}^n \rightarrow \mathbb{R}$ is a vector of quadratic functions. Hence,

$$q_i(x) = x^T Q^i x$$

where it is assumed that

$$Q^i = (Q^i)^T.$$

- $\mathcal{J} \subseteq \{1, 2, \ldots, n\}$ is an index set of the integer constrained variables.

16.1.1 File Sections

An LP formatted file contains a number of sections specifying the objective, constraints, variable bounds, and variable types. The section keywords may be any mix of upper and lower case letters.

Objective Function

The first section beginning with one of the keywords

```
max
maximum
maximize
min
minimum
minimize
```

defines the objective sense and the objective function, i.e.

$$c^T x + \frac{1}{2} x^T Q^o x.$$

The objective may be given a name by writing

```
myname:
```

before the expressions. If no name is given, then the objective is named `obj`.

The objective function contains linear and quadratic terms. The linear terms are written as

```
4 x1 + x2 - 0.1 x3
```

and so forth. The quadratic terms are written in square brackets (`[ ]/2`) and are either squared or multiplied as in the examples

```
x1^2
```

and

```
x1 * x2
```

There may be zero or more pairs of brackets containing quadratic expressions.

An example of an objective section is

```
minimize
myobj: 4 x1 + x2 - 0.1 x3 + [ x1^2 + 2.1 x1 * x2 ]/2
```

Please note that the quadratic expressions are multiplied with $\frac{1}{2}$, so that the above expression means

$$\text{minimize} \quad 4x_1 + x_2 - 0.1 \cdot x_3 + \tfrac{1}{2}(x_1^2 + 2.1 \cdot x_1 \cdot x_2)$$

If the same variable occurs more than once in the linear part, the coefficients are added, so that `4 x1 + 2 x1` is equivalent to `6 x1`. In the quadratic expressions `x1 * x2` is equivalent to `x2 * x1` and, as in the linear part, if the same variables multiplied or squared occur several times their coefficients are added.

Constraints

The second section beginning with one of the keywords

```
subj to
subject to
s.t.
st
```

defines the linear constraint matrix A and the quadratic matrices Q^i.

A constraint contains a name (optional), expressions adhering to the same rules as in the objective and a bound:

```
subject to
con1: x1 + x2 + [ x3^2 ]/2 <= 5.1
```

The bound type (here `<=`) may be any of `<`, `<=`, `=`, `>`, `>=` (`<` and `<=` mean the same), and the bound may be any number.

In the standard LP format it is not possible to define more than one bound per line, but **MOSEK** supports defining ranged constraints by using double-colon (`::`) instead of a single-colon (`:`) after the constraint name, i.e.

$$-5 \leq x_1 + x_2 \leq 5 \tag{16.1}$$

may be written as

```
con:: -5 < x_1 + x_2 < 5
```

By default **MOSEK** writes ranged constraints this way.

If the files must adhere to the LP standard, ranged constraints must either be split into upper bounded and lower bounded constraints or be written as an equality with a slack variable. For example the expression (16.1) may be written as

$$x_1 + x_2 - sl_1 = 0, \ -5 \leq sl_1 \leq 5.$$

Bounds

Bounds on the variables can be specified in the bound section beginning with one of the keywords

```
bound
bounds
```

The bounds section is optional but should, if present, follow the `subject to` section. All variables listed in the bounds section must occur in either the objective or a constraint.

The default lower and upper bounds are 0 and $+\infty$. A variable may be declared free with the keyword `free`, which means that the lower bound is $-\infty$ and the upper bound is $+\infty$. Furthermore it may be assigned a finite lower and upper bound. The bound definitions for a given variable may be written in one or two lines, and bounds can be any number or $\pm\infty$ (written as `+inf`/`-inf`/`+infinity`/`-infinity`) as in the example

```
bounds
x1 free
x2 <= 5
0.1 <= x2
x3 = 42
2 <= x4 < +inf
```

Variable Types

The final two sections are optional and must begin with one of the keywords

```
bin
binaries
binary
```

and

```
gen
general
```

Under `general` all integer variables are listed, and under `binary` all binary (integer variables with bounds 0 and 1) are listed:

```
general
x1 x2
binary
x3 x4
```

Again, all variables listed in the binary or general sections must occur in either the objective or a constraint.

Terminating Section

Finally, an LP formatted file must be terminated with the keyword

```
end
```

16.1.2 LP File Examples

Linear example `lo1.lp`

```
\ File: lo1.lp
maximize
obj: 3 x1 + x2 + 5 x3 + x4
subject to
c1:  3 x1 + x2 + 2 x3 = 30
c2:  2 x1 + x2 + 3 x3 + x4 >= 15
c3:  2 x2 + 3 x4 <= 25
bounds
 0 <= x1 <= +infinity
 0 <= x2 <= 10
 0 <= x3 <= +infinity
 0 <= x4 <= +infinity
end
```

Mixed integer example `milo1.lp`

```
maximize
obj: x1 + 6.4e-01 x2
subject to
c1:  5e+01 x1 + 3.1e+01 x2 <= 2.5e+02
c2:  3e+00 x1 - 2e+00 x2 >= -4e+00
bounds
 0 <= x1 <= +infinity
 0 <= x2 <= +infinity
general
 x1 x2
end
```

16.1.3 LP Format peculiarities

Comments

Anything on a line after a \ is ignored and is treated as a comment.

Names

A name for an objective, a constraint or a variable may contain the letters `a-z`, `A-Z`, the digits `0-9` and the characters

```
!"#$%&()/,.;?@_'`|~
```

The first character in a name must not be a number, a period or the letter `e` or `E`. Keywords must not be used as names.

MOSEK accepts any character as valid for names, except `\0`. A name that is not allowed in LP file will be changed and a warning will be issued.

The algorithm for making names LP valid works as follows: The name is interpreted as an `utf-8` string. For a Unicode character `c`:

- If `c==_` (underscore), the output is `__` (two underscores).
- If `c` is a valid LP name character, the output is just `c`.
- If `c` is another character in the ASCII range, the output is `_XX`, where `XX` is the hexadecimal code for the character.
- If `c` is a character in the range *127-65535*, the output is `_uXXXX`, where `XXXX` is the hexadecimal code for the character.

- If `c` is a character above 65535, the output is `_UXXXXXXXX`, where `XXXXXXXX` is the hexadecimal code for the character.

Invalid `utf-8` substrings are escaped as `_XX'`, and if a name starts with a period, `e` or `E`, that character is escaped as `_XX`.

Variable Bounds

Specifying several upper or lower bounds on one variable is possible but **MOSEK** uses only the tightest bounds. If a variable is fixed (with `=`), then it is considered the tightest bound.

MOSEK Extensions to the LP Format

Some optimization software packages employ a more strict definition of the LP format than the one used by **MOSEK**. The limitations imposed by the strict LP format are the following:

- Quadratic terms in the constraints are not allowed.
- Names can be only 16 characters long.
- Lines must not exceed 255 characters in length.

To get around some of the inconveniences converting from other problem formats, **MOSEK** allows lines to contain 1024 characters and names may have any length (shorter than the 1024 characters).

If an LP formatted file created by **MOSEK** should satisfy the strict definition, then the parameter *iparam.write_lp_strict_format* should be set; note, however, that some problems cannot be written correctly as a strict LP formatted file. For instance, all names are truncated to 16 characters and hence they may lose their uniqueness and change the problem.

Internally in **MOSEK** names may contain any (printable) character, many of which cannot be used in LP names. Setting the parameters *iparam.read_lp_quoted_names* and *iparam.write_lp_quoted_names* allows **MOSEK** to use quoted names. The first parameter tells **MOSEK** to remove quotes from quoted names e.g, `"x1"`, when reading LP formatted files. The second parameter tells **MOSEK** to put quotes around any semi-illegal name (names beginning with a number or a period) and fully illegal name (containing illegal characters). As double quote is a legal character in the LP format, quoting semi-illegal names makes them legal in the pure LP format as long as they are still shorter than 16 characters. Fully illegal names are still illegal in a pure LP file.

The strict LP format

The LP format is not a formal standard and different vendors have slightly different interpretations of the LP format. To make **MOSEK**'s definition of the LP format more compatible with the definitions of other vendors set the parameter *iparam.write_lp_strict_format* to *onoffkey.on*.

This setting may lead to truncation of some names and hence to an invalid LP file. The simple solution to this problem is to set the parameter *iparam.write_generic_names* to *onoffkey.on* which will cause all names to be renamed systematically in the output file.

Formatting of an LP File

A few parameters control the visual formatting of LP files written by **MOSEK** in order to make it easier to read the files. These parameters are

- *iparam.write_lp_line_width* sets the maximum number of characters on a single line. The default value is 80 corresponding roughly to the width of a standard text document.
- *iparam.write_lp_terms_per_line* sets the maximum number of terms per line; a term means a sign, a coefficient, and a name (for example `+ 42 elephants`). The default value is 0, meaning that there is no maximum.

Unnamed Constraints

Reading and writing an LP file with **MOSEK** may change it superficially. If an LP file contains unnamed constraints or objective these are given their generic names when the file is read (however unnamed constraints in **MOSEK** are written without names).

16.2 The MPS File Format

MOSEK supports the standard MPS format with some extensions. For a detailed description of the MPS format see the book by Nazareth *[Naz87]*.

16.2.1 MPS File Structure

The version of the MPS format supported by **MOSEK** allows specification of an optimization problem of the form

$$\begin{array}{lccccc} \text{maximize/minimize} & & c^T x + q_0(x) & & \\ l^c & \leq & Ax + q(x) & \leq & u^c, \\ l^x & \leq & x & \leq & u^x, \\ & & x \in \mathcal{K}, & & \\ & & x_{\mathcal{J}} \quad \text{integer}, & & \end{array} \tag{16.2}$$

where

- $x \in \mathbb{R}^n$ is the vector of decision variables.
- $A \in \mathbb{R}^{m \times n}$ is the constraint matrix.
- $l^c \in \mathbb{R}^m$ is the lower limit on the activity for the constraints.
- $u^c \in \mathbb{R}^m$ is the upper limit on the activity for the constraints.
- $l^x \in \mathbb{R}^n$ is the lower limit on the activity for the variables.
- $u^x \in \mathbb{R}^n$ is the upper limit on the activity for the variables.
- $q : \mathbb{R}^n \to \mathbb{R}$ is a vector of quadratic functions. Hence,

  $$q_i(x) = \frac{1}{2} x^T Q^i x$$

  where it is assumed that $Q^i = (Q^i)^T$. Please note the explicit $\frac{1}{2}$ in the quadratic term and that Q^i is required to be symmetric. The same applies to q_0.
- $\mathcal{K}$ is a convex cone.
- $\mathcal{J} \subseteq \{1, 2, \ldots, n\}$ is an index set of the integer-constrained variables.
- c is the vector of objective coefficients.

An MPS file with one row and one column can be illustrated like this:

```
*        1         2         3         4         5         6
*23456789012345678901234567890123456789012345678901234567890
NAME          [name]
OBJSENSE
    [objsense]
OBJNAME       [objname]
ROWS
 ?  [cname1]
COLUMNS
    [vname1]  [cname1]  [value1]       [cname2]  [value2]
RHS
    [name]    [cname1]  [value1]       [cname2]  [value2]
RANGES
    [name]    [cname1]  [value1]       [cname2]  [value2]
QSECTION      [cname1]
    [vname1]  [vname2]  [value1]       [vname3]  [value2]
QMATRIX
    [vname1]  [vname2]  [value1]
```

(continues on next page)

(continued from previous page)

```
QUADOBJ
    [vname1]  [vname2]  [value1]
QCMATRIX      [cname1]
    [vname1]  [vname2]  [value1]
BOUNDS
 ?? [name]    [vname1]  [value1]
CSECTION      [kname1]  [value1]       [ktype]
    [vname1]
ENDATA
```

Here the names in capitals are keywords of the MPS format and names in brackets are custom defined names or values. A couple of notes on the structure:

- Fields: All items surrounded by brackets appear in *fields*. The fields named "`valueN`" are numerical values. Hence, they must have the format

  ```
  [+|-]XXXXXXX.XXXXXX[[e|E][+|-]XXX]
  ```

  where

  ```
  X = [0|1|2|3|4|5|6|7|8|9].
  ```

- Sections: The MPS file consists of several sections where the names in capitals indicate the beginning of a new section. For example, `COLUMNS` denotes the beginning of the columns section.
- Comments: Lines starting with an `*` are comment lines and are ignored by **MOSEK**.
- Keys: The question marks represent keys to be specified later.
- Extensions: The sections `QSECTION` and `CSECTION` are specific **MOSEK** extensions of the MPS format. The sections `QMATRIX`, `QUADOBJ` and `QCMATRIX` are included for sake of compatibility with other vendors extensions to the MPS format.
- The standard MPS format is a fixed format, i.e. everything in the MPS file must be within certain fixed positions. **MOSEK** also supports a *free format*. See Sec. 16.2.5 for details.

Linear example `lo1.mps`

A concrete example of a MPS file is presented below:

```
* File: lo1.mps
NAME          lo1
OBJSENSE
    MAX
ROWS
 N  obj
 E  c1
 G  c2
 L  c3
COLUMNS
    x1        obj       3
    x1        c1        3
    x1        c2        2
    x2        obj       1
    x2        c1        1
    x2        c2        1
    x2        c3        2
    x3        obj       5
    x3        c1        2
    x3        c2        3
    x4        obj       1
    x4        c2        1
```

(continues on next page)

(continued from previous page)

```
    x4        c3        3
RHS
    rhs       c1        30
    rhs       c2        15
    rhs       c3        25
RANGES
BOUNDS
 UP bound     x2        10
ENDATA
```

Subsequently each individual section in the MPS format is discussed.

NAME (optional)

In this section a name ([name]) is assigned to the problem.

OBJSENSE (optional)

This is an optional section that can be used to specify the sense of the objective function. The OBJSENSE section contains one line at most which can be one of the following:

```
MIN
MINIMIZE
MAX
MAXIMIZE
```

It should be obvious what the implication is of each of these four lines.

OBJNAME (optional)

This is an optional section that can be used to specify the name of the row that is used as objective function. objname should be a valid row name.

ROWS

A record in the ROWS section has the form

```
?  [cname1]
```

where the requirements for the fields are as follows:

Field	Starting Position	Max Width	required	Description
?	2	1	Yes	Constraint key
[cname1]	5	8	Yes	Constraint name

Hence, in this section each constraint is assigned a unique name denoted by [cname1]. Please note that [cname1] starts in position 5 and the field can be at most 8 characters wide. An initial key ? must be present to specify the type of the constraint. The key can have values E, G, L, or N with the following interpretation:

Constraint type	l_i^c	u_i^c
E (equal)	finite	$= l_i^c$
G (greater)	finite	∞
L (lower)	$-\infty$	finite
N (none)	$-\infty$	∞

In the MPS format the objective vector is not specified explicitly, but one of the constraints having the key N will be used as the objective vector c . In general, if multiple N type constraints are specified, then the first will be used as the objective vector c, unless something else was specified in the section OBJNAME.

COLUMNS

In this section the elements of A are specified using one or more records having the form:

```
[vname1]  [cname1]    [value1]     [cname2]  [value2]
```

where the requirements for each field are as follows:

Field	Starting Position	Max Width	required	Description
[vname1]	5	8	Yes	Variable name
[cname1]	15	8	Yes	Constraint name
[value1]	25	12	Yes	Numerical value
[cname2]	40	8	No	Constraint name
[value2]	50	12	No	Numerical value

Hence, a record specifies one or two elements a_{ij} of A using the principle that [vname1] and [cname1] determines j and i respectively. Please note that [cname1] must be a constraint name specified in the ROWS section. Finally, [value1] denotes the numerical value of a_{ij} . Another optional element is specified by [cname2], and [value2] for the variable specified by [vname1]. Some important comments are:

- All elements belonging to one variable must be grouped together.
- Zero elements of A should not be specified.
- At least one element for each variable should be specified.

RHS (optional)

A record in this section has the format

```
[name]    [cname1]    [value1]     [cname2]  [value2]
```

where the requirements for each field are as follows:

Field	Starting Position	Max Width	required	Description
[name]	5	8	Yes	Name of the RHS vector
[cname1]	15	8	Yes	Constraint name
[value1]	25	12	Yes	Numerical value
[cname2]	40	8	No	Constraint name
[value2]	50	12	No	Numerical value

The interpretation of a record is that [name] is the name of the RHS vector to be specified. In general, several vectors can be specified. [cname1] denotes a constraint name previously specified in the ROWS section. Now, assume that this name has been assigned to the i -h constraint and v_1 denotes the value specified by [value1], then the interpretation of v_1 is:

Constraint	l_i^c	u_i^c
E	v_1	v_1
G	v_1	
L		v_1
N		

An optional second element is specified by [cname2] and [value2] and is interpreted in the same way. Please note that it is not necessary to specify zero elements, because elements are assumed to be zero.

RANGES (optional)

A record in this section has the form

```
[name]    [cname1]    [value1]     [cname2]  [value2]
```

where the requirements for each fields are as follows:

Field	Starting Position	Max Width	required	Description
`[name]`	5	8	Yes	Name of the `RANGE` vector
`[cname1]`	15	8	Yes	Constraint name
`[value1]`	25	12	Yes	Numerical value
`[cname2]`	40	8	No	Constraint name
`[value2]`	50	12	No	Numerical value

The records in this section are used to modify the bound vectors for the constraints, i.e. the values in l^c and u^c . A record has the following interpretation: `[name]` is the name of the `RANGE` vector and `[cname1]` is a valid constraint name. Assume that `[cname1]` is assigned to the i-th constraint and let v_1 be the value specified by `[value1]`, then a record has the interpretation:

Constraint type	Sign of v_1	l_i^c	u_i^c
`E`	$-$	$u_i^c + v_1$	
`E`	$+$		$l_i^c + v_1$
`G`	$-$ or $+$		$l_i^c + \lvert v_1 \rvert$
`L`	$-$ or $+$	$u_i^c - \lvert v_1 \rvert$	
`N`			

Another constraint bound can optionally be modified using `[cname2]` and `[value2]` the same way.

`QSECTION` (optional)

Within the `QSECTION` the label `[cname1]` must be a constraint name previously specified in the `ROWS` section. The label `[cname1]` denotes the constraint to which the quadratic terms belong. A record in the `QSECTION` has the form

```
[vname1]  [vname2]    [value1]     [vname3]  [value2]
```

where the requirements for each field are:

Field	Starting Position	Max Width	required	Description
`[vname1]`	5	8	Yes	Variable name
`[vname2]`	15	8	Yes	Variable name
`[value1]`	25	12	Yes	Numerical value
`[vname3]`	40	8	No	Variable name
`[value2]`	50	12	No	Numerical value

A record specifies one or two elements in the lower triangular part of the Q^i matrix where `[cname1]` specifies the i . Hence, if the names `[vname1]` and `[vname2]` have been assigned to the k-th and j-th variable, then Q^i_{kj} is assigned the value given by `[value1]` An optional second element is specified in the same way by the fields `[vname1]`, `[vname3]`, and `[value2]`.

The example

$$\begin{array}{lccc} \text{minimize} & -x_2 + \frac{1}{2}(2x_1^2 - 2x_1x_3 + 0.2x_2^2 + 2x_3^2) & & \\ \text{subject to} & x_1 + x_2 + x_3 & \geq & 1, \\ & x \geq 0 & & \end{array}$$

has the following MPS file representation

```
* File: qo1.mps
NAME          qo1
ROWS
 N  obj
 G  c1
COLUMNS
    x1        c1         1.0
    x2        obj        -1.0
    x2        c1         1.0
```

(continues on next page)

(continued from previous page)

```
    x3        c1        1.0
RHS
    rhs       c1        1.0
QSECTION      obj
    x1        x1        2.0
    x1        x3        -1.0
    x2        x2        0.2
    x3        x3        2.0
ENDATA
```

Regarding the QSECTIONs please note that:

- Only one QSECTION is allowed for each constraint.
- The QSECTIONs can appear in an arbitrary order after the COLUMNS section.
- All variable names occurring in the QSECTION must already be specified in the COLUMNS section.
- All entries specified in a QSECTION are assumed to belong to the lower triangular part of the quadratic term of Q .

QMATRIX/QUADOBJ **(optional)**

The QMATRIX and QUADOBJ sections allow to define the quadratic term of the objective function. They differ in how the quadratic term of the objective function is stored:

- QMATRIX stores all the nonzeros coefficients, without taking advantage of the symmetry of the Q matrix.
- QUADOBJ stores the upper diagonal nonzero elements of the Q matrix.

A record in both sections has the form:

```
[vname1]  [vname2]    [value1]
```

where the requirements for each field are:

Field	Starting Position	Max Width	required	Description
[vname1]	5	8	Yes	Variable name
[vname2]	15	8	Yes	Variable name
[value1]	25	12	Yes	Numerical value

A record specifies one elements of the Q matrix in the objective function . Hence, if the names [vname1] and [vname2] have been assigned to the k-th and j-th variable, then Q_{kj} is assigned the value given by [value1]. Note that a line must appear for each off-diagonal coefficient if using a QMATRIX section, while only one entry is required in a QUADOBJ section. The quadratic part of the objective function will be evaluated as $1/2x^TQx$.

The example

$$\begin{array}{lccc} \text{minimize} & -x_2 + \frac{1}{2}(2x_1^2 - 2x_1x_3 + 0.2x_2^2 + 2x_3^2) & & \\ \text{subject to} & x_1 + x_2 + x_3 & \geq & 1, \\ & x \geq 0 & & \end{array}$$

has the following MPS file representation using QMATRIX

```
* File: qo1_matrix.mps
NAME          qo1_qmatrix
ROWS
 N  obj
 G  c1
COLUMNS
    x1        c1        1.0
    x2        obj       -1.0
```

(continues on next page)

```
    x2        c1        1.0
    x3        c1        1.0
RHS
    rhs       c1        1.0
QMATRIX
    x1        x1        2.0
    x1        x3        -1.0
    x3        x1        -1.0
    x2        x2        0.2
    x3        x3        2.0
ENDATA
```

or the following using `QUADOBJ`

```
* File: qo1_quadobj.mps
NAME          qo1_quadobj
ROWS
 N  obj
 G  c1
COLUMNS
    x1        c1        1.0
    x2        obj       -1.0
    x2        c1        1.0
    x3        c1        1.0
RHS
    rhs       c1        1.0
QUADOBJ
    x1        x1        2.0
    x1        x3        -1.0
    x2        x2        0.2
    x3        x3        2.0
ENDATA
```

Please also note that:

- A `QMATRIX`/`QUADOBJ` section can appear in an arbitrary order after the `COLUMNS` section.
- All variable names occurring in the `QMATRIX`/`QUADOBJ` section must already be specified in the `COLUMNS` section.

`QCMATRIX` (optional)

A `QCMATRIX` section allows to specify the quadratic part of a given constraint. Within the `QCMATRIX` the label `[cname1]` must be a constraint name previously specified in the `ROWS` section. The label `[cname1]` denotes the constraint to which the quadratic term belongs. A record in the `QSECTION` has the form

```
[vname1]  [vname2]    [value1]
```

where the requirements for each field are:

Field	Starting Position	Max Width	required	Description
`[vname1]`	5	8	Yes	Variable name
`[vname2]`	15	8	Yes	Variable name
`[value1]`	25	12	Yes	Numerical value

A record specifies an entry of the Q^i matrix where `[cname1]` specifies the i. Hence, if the names `[vname1]` and `[vname2]` have been assigned to the k-th and j-th variable, then Q^i_{kj} is assigned the value given by `[value1]`. Moreover, the quadratic term is represented as $1/2x^TQx$.

The example

$$
\begin{array}{lccl}
\text{minimize} & x_2 & & \\
\text{subject to} & x_1+x_2+x_3 & \geq & 1, \\
 & \frac{1}{2}(-2x_1x_3+0.2x_2^2+2x_3^2) & \leq & 10, \\
 & x \geq 0 & &
\end{array}
$$

has the following MPS file representation

```
* File: qo1.mps
NAME          qo1
ROWS
 N  obj
 G  c1
 L  q1
COLUMNS
    x1        c1        1.0
    x2        obj       -1.0
    x2        c1        1.0
    x3        c1        1.0
RHS
    rhs       c1        1.0
    rhs       q1        10.0
QCMATRIX      q1
    x1        x1        2.0
    x1        x3        -1.0
    x3        x1        -1.0
    x2        x2        0.2
    x3        x3        2.0
ENDATA
```

Regarding the `QCMATRIX`s please note that:

- Only one `QCMATRIX` is allowed for each constraint.
- The `QCMATRIX`s can appear in an arbitrary order after the `COLUMNS` section.
- All variable names occurring in the `QSECTION` must already be specified in the `COLUMNS` section.
- `QCMATRIX` does not exploit the symmetry of Q: an off-diagonal entry (i, j) should appear twice.

`BOUNDS` (optional)

In the `BOUNDS` section changes to the default bounds vectors l^x and u^x are specified. The default bounds vectors are $l^x = 0$ and $u^x = \infty$. Moreover, it is possible to specify several sets of bound vectors. A record in this section has the form

```
?? [name]    [vname1]    [value1]
```

where the requirements for each field are:

Field	Starting Position	Max Width	Required	Description
`??`	2	2	Yes	Bound key
`[name]`	5	8	Yes	Name of the `BOUNDS` vector
`[vname1]`	15	8	Yes	Variable name
`[value1]`	25	12	No	Numerical value

Hence, a record in the `BOUNDS` section has the following interpretation: `[name]` is the name of the bound vector and `[vname1]` is the name of the variable for which the bounds are modified by the record. `??` and `[value1]` are used to modify the bound vectors according to the following table:

`??`	l_j^x	u_j^x	Made integer (added to $\mathcal{J}$)
`FR`	$-\infty$	∞	No
`FX`	v_1	v_1	No
`LO`	v_1	unchanged	No
`MI`	$-\infty$	unchanged	No
`PL`	unchanged	∞	No
`UP`	unchanged	v_1	No
`BV`	0	1	Yes
`LI`	$\lceil v_1 \rceil$	unchanged	Yes
`UI`	unchanged	$\lfloor v_1 \rfloor$	Yes

Here v_1 is the value specified by [value1].

CSECTION (optional)

The purpose of the CSECTION is to specify the conic constraint

$$x \in \mathcal{K}$$

in (16.2). It is assumed that $\mathcal{K}$ satisfies the following requirements. Let

$$x^t \in \mathbb{R}^{n^t}, \quad t = 1, \ldots, k$$

be vectors comprised of parts of the decision variables x so that each decision variable is a member of exactly **one** vector x^t, for example

$$x^1 = \begin{bmatrix} x_1 \\ x_4 \\ x_7 \end{bmatrix} \quad \text{and} \quad x^2 = \begin{bmatrix} x_6 \\ x_5 \\ x_3 \\ x_2 \end{bmatrix}.$$

Next define

$$\mathcal{K} := \left\{ x \in \mathbb{R}^n : \quad x^t \in \mathcal{K}_t, \quad t = 1, \ldots, k \right\}$$

where $\mathcal{K}_t$ must have one of the following forms:

- $\mathbb{R}$ set:

$$\mathcal{K}_t = \mathbb{R}^{n^t}.$$

- Zero cone:

$$\mathcal{K}_t = \{0\} \subseteq \mathbb{R}^{n^t}. \tag{16.3}$$

- Quadratic cone:

$$\mathcal{K}_t = \left\{ x \in \mathbb{R}^{n^t} : x_1 \geq \sqrt{\sum_{j=2}^{n^t} x_j^2} \right\}. \tag{16.4}$$

- Rotated quadratic cone:

$$\mathcal{K}_t = \left\{ x \in \mathbb{R}^{n^t} : 2x_1 x_2 \geq \sum_{j=3}^{n^t} x_j^2, \quad x_1, x_2 \geq 0 \right\}. \tag{16.5}$$

- Primal exponential cone:

$$\mathcal{K}_t = \left\{ x \in \mathbb{R}^3 : x_1 \geq x_2 \exp(x_3/x_2), \quad x_1, x_2 \geq 0 \right\}. \tag{16.6}$$

- Primal power cone (with parameter $0 < \alpha < 1$):

$$\mathcal{K}_t = \left\{ x \in \mathbb{R}^{n^t} : x_1^{\alpha} x_2^{1-\alpha} \geq \sqrt{\sum_{j=3}^{n^t} x_j^2}, \quad x_1, x_2 \geq 0 \right\}. \tag{16.7}$$

- Dual exponential cone:

$$\mathcal{K}_t = \left\{ x \in \mathbb{R}^3 : x_1 \geq -x_3 e^{-1} \exp(x_2/x_3), \quad x_3 \leq 0, x_1 \geq 0 \right\}. \tag{16.8}$$

- Dual power cone (with parameter $0 < \alpha < 1$):

$$\mathcal{K}_t = \left\{ x \in \mathbb{R}^{n^t} : \left(\frac{x_1}{\alpha}\right)^{\alpha} \left(\frac{x_2}{1-\alpha}\right)^{1-\alpha} \geq \sqrt{\sum_{j=3}^{n^t} x_j^2}, \quad x_1, x_2 \geq 0 \right\}. \tag{16.9}$$

In general, membership in the $\mathbb{R}$ set is not specified. If a variable is not a member of any other cone then it is assumed to be a member of the $\mathbb{R}$ cone.

Next, let us study an example. Assume that the power cone

$$x_4^{1/3} x_5^{2/3} \geq |x_8|$$

and the rotated quadratic cone

$$2x_3x_7 \geq x_1^2 + x_0^2, \quad x_3, x_7 \geq 0,$$

should be specified in the MPS file. One `CSECTION` is required for each cone and they are specified as follows:

```
*        1         2         3         4         5         6
*234567890123456789012345678901234567890123456789012345678901234567890
CSECTION      konea     3e-1           PPOW
x4
x5
x8
CSECTION      koneb     0.0            RQUAD
x7
x3
x1
x0
```

In general, a `CSECTION` header has the format

```
CSECTION      [kname1]     [value1]      [ktype]
```

where the requirements for each field are as follows:

Field	Starting Position	Max Width	Required	Description
`[kname1]`	15	8	Yes	Name of the cone
`[value1]`	25	12	No	Cone parameter
`[ktype]`	40		Yes	Type of the cone.

The possible cone type keys are:

`[ktype]`	Members	`[value1]`	Interpretation.
`ZERO`	≥ 0	unused	Zero cone (16.3).
`QUAD`	≥ 1	unused	Quadratic cone (16.4).
`RQUAD`	≥ 2	unused	Rotated quadratic cone (16.5).
`PEXP`	3	unused	Primal exponential cone (16.6).
`PPOW`	≥ 2	α	Primal power cone (16.7).
`DEXP`	3	unused	Dual exponential cone (16.8).
`DPOW`	≥ 2	α	Dual power cone (16.9).

A record in the `CSECTION` has the format

```
[vname1]
```

where the requirements for each field are

Field	Starting Position	Max Width	required	Description
`[vname1]`	5	8	Yes	A valid variable name

A variable must occur in at most one `CSECTION`.

ENDATA

This keyword denotes the end of the MPS file.

16.2.2 Integer Variables

Using special bound keys in the BOUNDS section it is possible to specify that some or all of the variables should be integer-constrained i.e. be members of $\mathcal{J}$. However, an alternative method is available. This method is available only for backward compatibility and we recommend that it is not used. This method requires that markers are placed in the COLUMNS section as in the example:

```
COLUMNS
x1        obj       -10.0          c1        0.7
x1        c2        0.5            c3        1.0
x1        c4        0.1
* Start of integer-constrained variables.
MARK000   'MARKER'                 'INTORG'
x2        obj       -9.0           c1        1.0
x2        c2        0.8333333333   c3        0.66666667
x2        c4        0.25
x3        obj       1.0            c6        2.0
MARK001   'MARKER'                 'INTEND'
* End of integer-constrained variables.
```

Please note that special marker lines are used to indicate the start and the end of the integer variables. Furthermore be aware of the following

- All variables between the markers are assigned a default lower bound of 0 and a default upper bound of 1. **This may not be what is intended.** If it is not intended, the correct bounds should be defined in the BOUNDS section of the MPS formatted file.
- **MOSEK** ignores field 1, i.e. MARK0001 and MARK001, however, other optimization systems require them.
- Field 2, i.e. MARKER, must be specified including the single quotes. This implies that no row can be assigned the name MARKER.
- Field 3 is ignored and should be left blank.
- Field 4, i.e. INTORG and INTEND, must be specified.
- It is possible to specify several such integer marker sections within the COLUMNS section.

16.2.3 General Limitations

- An MPS file should be an ASCII file.

16.2.4 Interpretation of the MPS Format

Several issues related to the MPS format are not well-defined by the industry standard. However, **MOSEK** uses the following interpretation:

- If a matrix element in the COLUMNS section is specified multiple times, then the multiple entries are added together.
- If a matrix element in a QSECTION section is specified multiple times, then the multiple entries are added together.

16.2.5 The Free MPS Format

MOSEK supports a free format variation of the MPS format. The free format is similar to the MPS file format but less restrictive, e.g. it allows longer names. However, a name must not contain any blanks.

Moreover, by default a line in the MPS file must not contain more than 1024 characters. By modifying the parameter *iparam.read_mps_width* an arbitrary large line width will be accepted.

The free MPS format is default. To change to the strict and other formats use the parameter *iparam.read_mps_format*.

16.3 The OPF Format

The *Optimization Problem Format (OPF)* is an alternative to LP and MPS files for specifying optimization problems. It is row-oriented, inspired by the CPLEX LP format.

Apart from containing objective, constraints, bounds etc. it may contain complete or partial solutions, comments and extra information relevant for solving the problem. It is designed to be easily read and modified by hand and to be forward compatible with possible future extensions.

Intended use

The OPF file format is meant to replace several other files:

- The LP file format: Any problem that can be written as an LP file can be written as an OPF file too; furthermore it naturally accommodates ranged constraints and variables as well as arbitrary characters in names, fixed expressions in the objective, empty constraints, and conic constraints.
- Parameter files: It is possible to specify integer, double and string parameters along with the problem (or in a separate OPF file).
- Solution files: It is possible to store a full or a partial solution in an OPF file and later reload it.

16.3.1 The File Format

The format uses tags to structure data. A simple example with the basic sections may look like this:

```
[comment]
This is a comment. You may write almost anything here...
[/comment]

# This is a single-line comment.

[objective min 'myobj']
x + 3 y + x^2 + 3 y^2 + z + 1
[/objective]

[constraints]
[con 'con01'] 4 <= x + y  [/con]
[/constraints]

[bounds]
[b] -10 <= x,y <= 10  [/b]

[cone quad] x,y,z [/cone]
[/bounds]
```

A scope is opened by a tag of the form `[tag]` and closed by a tag of the form `[/tag]`. An opening tag may accept a list of unnamed and named arguments, for examples:

```
[tag value] tag with one unnamed argument [/tag]
[tag arg=value] tag with one named argument [/tag]
```

Unnamed arguments are identified by their order, while named arguments may appear in any order, but never before an unnamed argument. The `value` can be a quoted, single-quoted or double-quoted text string, i.e.

```
[tag 'value']     single-quoted value [/tag]
[tag arg='value'] single-quoted value [/tag]
[tag "value"]     double-quoted value [/tag]
[tag arg="value"] double-quoted value [/tag]
```

16.3.2 Sections

The recognized tags are

[comment]

A comment section. This can contain *almost* any text: Between single quotes (') or double quotes (") any text may appear. Outside quotes the markup characters ([and]) must be prefixed by backslashes. Both single and double quotes may appear alone or inside a pair of quotes if it is prefixed by a backslash.

[objective]

The objective function: This accepts one or two parameters, where the first one (in the above example `min`) is either `min` or `max` (regardless of case) and defines the objective sense, and the second one (above `myobj`), if present, is the objective name. The section may contain linear and quadratic expressions.

If several objectives are specified, all but the last are ignored.

[constraints]

This does not directly contain any data, but may contain subsections `con` defining a linear constraint.

[con]

Defines a single constraint; if an argument is present (`[con NAME]`) this is used as the name of the constraint, otherwise it is given a null-name. The section contains a constraint definition written as linear and quadratic expressions with a lower bound, an upper bound, with both or with an equality. Examples:

```
[constraints]
[con 'con1'] 0 <= x + y       [/con]
[con 'con2'] 0 >= x + y       [/con]
[con 'con3'] 0 <= x + y <= 10 [/con]
[con 'con4']      x + y  = 10 [/con]
[/constraints]
```

Constraint names are unique. If a constraint is specified which has the same name as a previously defined constraint, the new constraint replaces the existing one.

[bounds]

This does not directly contain any data, but may contain subsections `b` (linear bounds on variables) and `cone` (cones).

[b]

Bound definition on one or several variables separated by comma (`,`). An upper or lower bound on a variable replaces any earlier defined bound on that variable. If only one bound (upper or lower) is given only this bound is replaced. This means that upper and lower bounds can be specified separately. So the OPF bound definition:

```
[b]  x,y >= -10  [/b]
[b]  x,y <= 10   [/b]
```

results in the bound $-10 \leq x, y \leq 10$.

[cone]

Specifies a cone. A cone is defined as a sequence of variables which belong to a single unique cone. The supported cone types are:

- quad: a quadratic cone of n variables $x_1, \ldots, x_n$ defines a constraint of the form

$$x_1^2 \geq \sum_{i=2}^{n} x_i^2, \quad x_1 \geq 0.$$

- rquad: a rotated quadratic cone of n variables $x_1, \ldots, x_n$ defines a constraint of the form

$$2x_1x_2 \geq \sum_{i=3}^{n} x_i^2, \quad x_1, x_2 \geq 0.$$

- pexp: primal exponential cone of 3 variables x_1, x_2, x_3 defines a constraint of the form

$$x_1 \geq x_2 \exp(x_3/x_2), \quad x_1, x_2 \geq 0.$$

- ppow with parameter $0 < \alpha < 1$: primal power cone of n variables $x_1, \ldots, x_n$ defines a constraint of the form

$$x_1^{\alpha} x_2^{1-\alpha} \geq \sqrt{\sum_{j=3}^{n} x_j^2}, \quad x_1, x_2 \geq 0.$$

- dexp: dual exponential cone of 3 variables x_1, x_2, x_3 defines a constraint of the form

$$x_1 \geq -x_3 e^{-1} \exp(x_2/x_3), \quad x_3 \leq 0, x_1 \geq 0.$$

- dpow with parameter $0 < \alpha < 1$: dual power cone of n variables $x_1, \ldots, x_n$ defines a constraint of the form

$$\left(\frac{x_1}{\alpha}\right)^{\alpha} \left(\frac{x_2}{1-\alpha}\right)^{1-\alpha} \geq \sqrt{\sum_{j=3}^{n^t} x_j^2}, \quad x_1, x_2 \geq 0.$$

- zero: zero cone of n variables $x_1, \ldots, x_n$ defines a constraint of the form

$$x_1 = \cdots = x_n = 0$$

A [bounds]-section example:

```
[bounds]
[b]  0 <= x,y <= 10  [/b] # ranged bound
[b] 10 >= x,y >=  0  [/b] # ranged bound
[b]  0 <= x,y <= inf [/b] # using inf
[b]       x,y free   [/b] # free variables
# Let (x,y,z,w) belong to the cone K
[cone rquad] x,y,z,w  [/cone] # rotated quadratic cone
[cone ppow '3e-01' 'a'] x1, x2, x3 [/cone] # power cone with alpha=1/3 and name 'a'
[/bounds]
```

By default all variables are free.

[variables]

This defines an ordering of variables as they should appear in the problem. This is simply a space-separated list of variable names.

[integer]

This contains a space-separated list of variables and defines the constraint that the listed variables must be integer-valued.

[hints]

This may contain only non-essential data; for example estimates of the number of variables, constraints and non-zeros. Placed before all other sections containing data this may reduce the time spent reading the file.

In the `hints` section, any subsection which is not recognized by **MOSEK** is simply ignored. In this section a hint is defined as follows:

```
[hint ITEM] value [/hint]
```

The hints recognized by **MOSEK** are:

- `numvar` (number of variables),
- `numcon` (number of linear/quadratic constraints),
- `numanz` (number of linear non-zeros in constraints),
- `numqnz` (number of quadratic non-zeros in constraints).

[solutions]

This section can contain a set of full or partial solutions to a problem. Each solution must be specified using a `[solution]`-section, i.e.

```
[solutions]
[solution]...[/solution] #solution 1
[solution]...[/solution] #solution 2
#other solutions....
[solution]...[/solution] #solution n
[/solutions]
```

The syntax of a `[solution]`-section is the following:

```
[solution SOLTYPE status=STATUS]...[/solution]
```

where `SOLTYPE` is one of the strings

- `interior`, a non-basic solution,
- `basic`, a basic solution,
- `integer`, an integer solution,

and `STATUS` is one of the strings

- `UNKNOWN`,
- `OPTIMAL`,
- `INTEGER_OPTIMAL`,
- `PRIM_FEAS`,
- `DUAL_FEAS`,
- `PRIM_AND_DUAL_FEAS`,
- `NEAR_OPTIMAL`,
- `NEAR_PRIM_FEAS`,

- NEAR_DUAL_FEAS,
- NEAR_PRIM_AND_DUAL_FEAS,
- PRIM_INFEAS_CER,
- DUAL_INFEAS_CER,
- NEAR_PRIM_INFEAS_CER,
- NEAR_DUAL_INFEAS_CER,
- NEAR_INTEGER_OPTIMAL.

Most of these values are irrelevant for input solutions; when constructing a solution for simplex hot-start or an initial solution for a mixed integer problem the safe setting is UNKNOWN.

A [solution]-section contains [con] and [var] sections. Each [con] and [var] section defines solution information for a single variable or constraint, specified as list of KEYWORD/value pairs, in any order, written as

```
KEYWORD=value
```

Allowed keywords are as follows:

- sk. The status of the item, where the value is one of the following strings:
  - LOW, the item is on its lower bound.
  - UPR, the item is on its upper bound.
  - FIX, it is a fixed item.
  - BAS, the item is in the basis.
  - SUPBAS, the item is super basic.
  - UNK, the status is unknown.
  - INF, the item is outside its bounds (infeasible).
- lvl Defines the level of the item.
- sl Defines the level of the dual variable associated with its lower bound.
- su Defines the level of the dual variable associated with its upper bound.
- sn Defines the level of the variable associated with its cone.
- y Defines the level of the corresponding dual variable (for constraints only).

A [var] section should always contain the items sk, lvl, sl and su. Items sl and su are not required for integer solutions.

A [con] section should always contain sk, lvl, sl, su and y.

An example of a solution section

```
[solution basic status=UNKNOWN]
[var x0] sk=LOW    lvl=5.0      [/var]
[var x1] sk=UPR    lvl=10.0     [/var]
[var x2] sk=SUPBAS lvl=2.0  sl=1.5 su=0.0 [/var]

[con c0] sk=LOW    lvl=3.0 y=0.0 [/con]
[con c0] sk=UPR    lvl=0.0 y=5.0 [/con]
[/solution]
```

- [vendor] This contains solver/vendor specific data. It accepts one argument, which is a vendor ID – for **MOSEK** the ID is simply mosek – and the section contains the subsection parameters defining solver parameters. When reading a vendor section, any unknown vendor can be safely ignored. This is described later.

Comments using the # may appear anywhere in the file. Between the # and the following line-break any text may be written, including markup characters.

16.3.3 Numbers

Numbers, when used for parameter values or coefficients, are written in the usual way by the `printf` function. That is, they may be prefixed by a sign (`+` or `-`) and may contain an integer part, decimal part and an exponent. The decimal point is always `.` (a dot). Some examples are

```
1
1.0
.0
1.
1e10
1e+10
1e-10
```

Some *invalid* examples are

```
e10   # invalid, must contain either integer or decimal part
.     # invalid
.e10  # invalid
```

More formally, the following standard regular expression describes numbers as used:

```
[+|-]?([0-9]+[.][0-9]*|[.][0-9]+)([eE][+|-]?[0-9]+)?
```

16.3.4 Names

Variable names, constraint names and objective name may contain arbitrary characters, which in some cases must be enclosed by quotes (single or double) that in turn must be preceded by a backslash. Unquoted names must begin with a letter (`a-z` or `A-Z`) and contain only the following characters: the letters `a-z` and `A-Z`, the digits `0-9`, braces (`{` and `}`) and underscore (`_`).

Some examples of legal names:

```
an_unquoted_name
another_name{123}
'single quoted name'
"double quoted name"
"name with \\"quote\\" in it"
"name with []s in it"
```

16.3.5 Parameters Section

In the `vendor` section solver parameters are defined inside the `parameters` subsection. Each parameter is written as

```
[p PARAMETER_NAME] value [/p]
```

where `PARAMETER_NAME` is replaced by a **MOSEK** parameter name, usually of the form `MSK_IPAR_...`, `MSK_DPAR_...` or `MSK_SPAR_...`, and the `value` is replaced by the value of that parameter; both integer values and named values may be used. Some simple examples are

```
[vendor mosek]
[parameters]
[p MSK_IPAR_OPF_MAX_TERMS_PER_LINE] 10     [/p]
[p MSK_IPAR_OPF_WRITE_PARAMETERS]   MSK_ON [/p]
[p MSK_DPAR_DATA_TOL_BOUND_INF]     1.0e18 [/p]
[/parameters]
[/vendor]
```

16.3.6 Writing OPF Files from MOSEK

To write an OPF file then make sure the file extension is .opf.

Then modify the following parameters to define what the file should contain:

iparam.opf_write_sol_bas	Include basic solution, if defined.
iparam.opf_write_sol_itg	Include integer solution, if defined.
iparam.opf_write_sol_itr	Include interior solution, if defined.
iparam.opf_write_solutions	Include solutions if they are defined. If this is off, no solutions are included.
iparam.opf_write_header	Include a small header with comments.
iparam.opf_write_problem	Include the problem itself — objective, constraints and bounds.
iparam.opf_write_parameters	Include all parameter settings.
iparam.opf_write_hints	Include hints about the size of the problem.

16.3.7 Examples

This section contains a set of small examples written in OPF and describing how to formulate linear, quadratic and conic problems.

Linear Example lo1.opf

Consider the example:

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

having the bounds

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

In the OPF format the example is displayed as shown in Listing 16.1.

Listing 16.1: Example of an OPF file for a linear problem.

```
[comment]
  The lo1 example in OPF format
[/comment]

[hints]
  [hint NUMVAR] 4 [/hint]
  [hint NUMCON] 3 [/hint]
  [hint NUMANZ] 9 [/hint]
[/hints]

[variables disallow_new_variables]
  x1 x2 x3 x4
[/variables]

[objective maximize 'obj']
   3 x1 + x2 + 5 x3 + x4
[/objective]

[constraints]
  [con 'c1'] 3 x1 +   x2 + 2 x3         = 30 [/con]
  [con 'c2'] 2 x1 +   x2 + 3 x3 +   x4 >= 15 [/con]
  [con 'c3']        2 x2        + 3 x4 <= 25 [/con]
[/constraints]
```

(continues on next page)

(continued from previous page)

```
[bounds]
  [b] 0 <= * [/b]
  [b] 0 <= x2 <= 10 [/b]
[/bounds]
```

Quadratic Example qo1.opf

An example of a quadratic optimization problem is

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

This can be formulated in `opf` as shown below.

Listing 16.2: Example of an OPF file for a quadratic problem.

```
[comment]
  The qo1 example in OPF format
[/comment]

[hints]
  [hint NUMVAR] 3 [/hint]
  [hint NUMCON] 1 [/hint]
  [hint NUMANZ] 3 [/hint]
  [hint NUMQNZ] 4 [/hint]
[/hints]

[variables disallow_new_variables]
  x1 x2 x3
[/variables]

[objective minimize 'obj']
  # The quadratic terms are often written with a factor of 1/2 as here,
  # but this is not required.

   - x2 + 0.5 ( 2.0 x1 ^ 2 - 2.0 x3 * x1 + 0.2 x2 ^ 2 + 2.0 x3 ^ 2 )
[/objective]

[constraints]
  [con 'c1'] 1.0 <= x1 + x2 + x3 [/con]
[/constraints]

[bounds]
  [b] 0 <= * [/b]
[/bounds]
```

Conic Quadratic Example cqo1.opf

Consider the example:

$$\begin{array}{lccc} \text{minimize} & x_3 + x_4 + x_5 & & \\ \text{subject to} & x_0 + x_1 + 2x_2 & = & 1, \\ & x_0, x_1, x_2 & \geq & 0, \\ & x_3 \geq \sqrt{x_0^2 + x_1^2}, & & \\ & 2x_4x_5 \geq x_2^2. & & \end{array}$$

Please note that the type of the cones is defined by the parameter to `[cone ...]`; the content of the `cone`-section is the names of variables that belong to the cone. The resulting OPF file is in Listing 16.3.

Listing 16.3: Example of an OPF file for a conic quadratic problem.

```
[comment]
  The cqo1 example in OPF format.
[/comment]

[hints]
  [hint NUMVAR] 6 [/hint]
  [hint NUMCON] 1 [/hint]
  [hint NUMANZ] 3 [/hint]
[/hints]

[variables disallow_new_variables]
  x1 x2 x3 x4 x5 x6
[/variables]

[objective minimize 'obj']
   x4 + x5 + x6
[/objective]

[constraints]
  [con 'c1']  x1 + x2 + 2e+00 x3 = 1e+00 [/con]
[/constraints]

[bounds]
  # We let all variables default to the positive orthant
  [b] 0 <= * [/b]

  # ...and change those that differ from the default
  [b] x4,x5,x6 free [/b]

  # Define quadratic cone: x4 >= sqrt( x1^2 + x2^2 )
  [cone quad 'k1'] x4, x1, x2 [/cone]

  # Define rotated quadratic cone: 2 x5 x6 >= x3^2
  [cone rquad 'k2'] x5, x6, x3 [/cone]
[/bounds]
```

Mixed Integer Example `milo1.opf`

Consider the mixed integer problem:

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

This can be implemented in `OPF` with the file in Listing 16.4.

Listing 16.4: Example of an OPF file for a mixed-integer linear problem.

```
[comment]
  The milo1 example in OPF format
[/comment]

[hints]
  [hint NUMVAR] 2 [/hint]
  [hint NUMCON] 2 [/hint]
  [hint NUMANZ] 4 [/hint]
[/hints]
```

(continues on next page)

(continued from previous page)

```
[variables disallow_new_variables]
  x1 x2
[/variables]

[objective maximize 'obj']
   x1 + 6.4e-1 x2
[/objective]

[constraints]
  [con 'c1'] 5e+1 x1 + 3.1e+1 x2 <= 2.5e+2 [/con]
  [con 'c2'] -4 <= 3 x1 - 2 x2 [/con]
[/constraints]

[bounds]
  [b] 0 <= * [/b]
[/bounds]

[integer]
  x1 x2
[/integer]
```

16.4 The CBF Format

This document constitutes the technical reference manual of the *Conic Benchmark Format* with file extension: `.cbf` or `.CBF`. It unifies linear, second-order cone (also known as conic quadratic) and semidefinite optimization with mixed-integer variables. The format has been designed with benchmark libraries in mind, and therefore focuses on compact and easily parsable representations. The problem structure is separated from the problem data, and the format moreover facilitates benchmarking of hotstart capability through sequences of changes.

16.4.1 How Instances Are Specified

This section defines the spectrum of conic optimization problems that can be formulated in terms of the keywords of the CBF format.

In the CBF format, conic optimization problems are considered in the following form:

$$\begin{array}{lll} \min / \max & g^{obj} & \\ & g_i \in \mathcal{K}_i, & i \in \mathcal{I}, \\ \text{s.t.} & G_i \in \mathcal{K}_i, & i \in \mathcal{I}^{PSD}, \\ & x_j \in \mathcal{K}_j, & j \in \mathcal{J}, \\ & \overline{X}_j \in \mathcal{K}_j, & j \in \mathcal{J}^{PSD}. \end{array} \tag{16.10}$$

- **Variables** are either scalar variables, x_j for $j \in \mathcal{J}$, or variables, $\overline{X}_j$ for $j \in \mathcal{J}^{PSD}$. Scalar variables can also be declared as integer.
- **Constraints** are affine expressions of the variables, either scalar-valued g_i for $i \in \mathcal{I}$, or matrix-valued G_i for $i \in \mathcal{I}^{PSD}$

$$g_i = \sum_{j \in \mathcal{J}^{PSD}} \langle F_{ij}, X_j \rangle + \sum_{j \in \mathcal{J}} a_{ij} x_j + b_i,$$

$$G_i = \sum_{j \in \mathcal{J}} x_j H_{ij} + D_i.$$

- The **objective function** is a scalar-valued affine expression of the variables, either to be minimized or maximized. We refer to this expression as g^{obj}

$$g^{obj} = \sum_{j \in \mathcal{J}^{PSD}} \langle F_j^{obj}, X_j \rangle + \sum_{j \in \mathcal{J}} a_j^{obj} x_j + b^{obj}.$$

CBF format can represent the following cones $\mathcal{K}$:

- **Free domain** - A cone in the linear family defined by

$$\{x \in \mathbb{R}^n\}, \text{ for } n \geq 1.$$

- **Positive orthant** - A cone in the linear family defined by

$$\{x \in \mathbb{R}^n \mid x_j \geq 0 \text{ for } j = 1, \ldots, n\}, \text{ for } n \geq 1.$$

- **Negative orthant** - A cone in the linear family defined by

$$\{x \in \mathbb{R}^n \mid x_j \leq 0 \text{ for } j = 1, \ldots, n\}, \text{ for } n \geq 1.$$

- **Fixpoint zero** - A cone in the linear family defined by

$$\{x \in \mathbb{R}^n \mid x_j = 0 \text{ for } j = 1, \ldots, n\}, \text{ for } n \geq 1.$$

- **Quadratic cone** - A cone in the second-order cone family defined by

$$\left\{ \begin{pmatrix} p \\ x \end{pmatrix} \in \mathbb{R} \times \mathbb{R}^{n-1},\ p^2 \geq x^T x,\ p \geq 0 \right\}, \text{ for } n \geq 2.$$

- **Rotated quadratic cone** - A cone in the second-order cone family defined by

$$\left\{ \begin{pmatrix} p \\ q \\ x \end{pmatrix} \in \mathbb{R} \times \mathbb{R} \times \mathbb{R}^{n-2},\ 2pq \geq x^T x,\ p \geq 0,\ q \geq 0 \right\}, \text{ for } n \geq 3.$$

16.4.2 The Structure of CBF Files

This section defines how information is written in the CBF format, without being specific about the type of information being communicated.

All information items belong to exactly one of the three groups of information. These information groups, and the order they must appear in, are:

1. File format.
2. Problem structure.
3. Problem data.

The first group, file format, provides information on how to interpret the file. The second group, problem structure, provides the information needed to deduce the type and size of the problem instance. Finally, the third group, problem data, specifies the coefficients and constants of the problem instance.

Information items

The format is composed as a list of information items. The first line of an information item is the `KEYWORD`, revealing the type of information provided. The second line - of some keywords only - is the `HEADER`, typically revealing the size of information that follows. The remaining lines are the `BODY` holding the actual information to be specified.

```
KEYWORD
BODY

KEYWORD
HEADER
BODY
```

The `KEYWORD` determines how each line in the `HEADER` and `BODY` is structured. Moreover, the number of lines in the `BODY` follows either from the `KEYWORD`, the `HEADER`, or from another information item required to precede it.

Embedded hotstart-sequences

A sequence of problem instances, based on the same problem structure, is within a single file. This is facilitated via the `CHANGE` within the problem data information group, as a separator between the information items of each instance. The information items following a `CHANGE` keyword are appending to, or changing (e.g., setting coefficients back to their default value of zero), the problem data of the preceding instance.

The sequence is intended for benchmarking of hotstart capability, where the solvers can reuse their internal state and solution (subject to the achieved accuracy) as warmpoint for the succeeding instance. Whenever this feature is unsupported or undesired, the keyword `CHANGE` should be interpreted as the end of file.

File encoding and line width restrictions

The format is based on the US-ASCII printable character set with two extensions as listed below. Note, by definition, that none of these extensions can be misinterpreted as printable US-ASCII characters:

- A line feed marks the end of a line, carriage returns are ignored.
- Comment-lines may contain unicode characters in UTF-8 encoding.

The line width is restricted to 512 bytes, with 3 bytes reserved for the potential carriage return, line feed and null-terminator.

Integers and floating point numbers must follow the ISO C decimal string representation in the standard C locale. The format does not impose restrictions on the magnitude of, or number of significant digits in numeric data, but the use of 64-bit integers and 64-bit IEEE 754 floating point numbers should be sufficient to avoid loss of precision.

Comment-line and whitespace rules

The format allows single-line comments respecting the following rule:

- Lines having first byte equal to '`#`' (US-ASCII 35) are comments, and should be ignored. Comments are only allowed between information items.

Given that a line is not a comment-line, whitespace characters should be handled according to the following rules:

- Leading and trailing whitespace characters should be ignored.
  - The seperator between multiple pieces of information on one line, is either one or more whitespace characters.
- Lines containing only whitespace characters are empty, and should be ignored. Empty lines are only allowed between information items.

16.4.3 Problem Specification

The problem structure

The problem structure defines the objective sense, whether it is minimization and maximization. It also defines the index sets, $\mathcal{J}$, $\mathcal{J}^{PSD}$, $\mathcal{I}$ and $\mathcal{I}^{PSD}$, which are all numbered from zero, $\{0, 1, \ldots\}$, and empty until explicitly constructed.

- **Scalar variables** are constructed in vectors restricted to a conic domain, such as $(x_0, x_1) \in \mathbb{R}^2_+$, $(x_2, x_3, x_4) \in \mathcal{Q}^3$, etc. In terms of the Cartesian product, this generalizes to

$$x \in \mathcal{K}_1^{n_1} \times \mathcal{K}_2^{n_2} \times \cdots \times \mathcal{K}_k^{n_k}$$

  which in the CBF format becomes:

```
VAR
n k
K1 n1
K2 n2
...
Kk nk
```

  where $\sum_i n_i = n$ is the total number of scalar variables. The list of supported cones is found in Table 16.3. Integrality of scalar variables can be specified afterwards.

- **PSD variables** are constructed one-by-one. That is, $X_j \succeq \mathbf{0}^{n_j \times n_j}$ for $j \in \mathcal{J}^{PSD}$, constructs a matrix-valued variable of size $n_j \times n_j$ restricted to be symmetric positive semidefinite. In the CBF format, this list of constructions becomes:

```
PSDVAR
N
n1
n2
...
nN
```

  where N is the total number of PSD variables.

- **Scalar constraints** are constructed in vectors restricted to a conic domain, such as $(g_0, g_1) \in \mathbb{R}^2_+$, $(g_2, g_3, g_4) \in \mathcal{Q}^3$, etc. In terms of the Cartesian product, this generalizes to

$$g \in \mathcal{K}_1^{m_1} \times \mathcal{K}_2^{m_2} \times \cdots \times \mathcal{K}_k^{m_k}$$

  which in the CBF format becomes:

```
CON
m k
K1 m1
K2 m2
..
Kk mk
```

  where $\sum_i m_i = m$ is the total number of scalar constraints. The list of supported cones is found in Table 16.3.

- **PSD constraints** are constructed one-by-one. That is, $G_i \succeq \mathbf{0}^{m_i \times m_i}$ for $i \in \mathcal{I}^{PSD}$, constructs a matrix-valued affine expressions of size $m_i \times m_i$ restricted to be symmetric positive semidefinite. In the CBF format, this list of constructions becomes

```
PSDCON
M
m1
m2
..
mM
```

where M is the total number of PSD constraints.

With the objective sense, variables (with integer indications) and constraints, the definitions of the many affine expressions follow in problem data.

Problem data

The problem data defines the coefficients and constants of the affine expressions of the problem instance. These are considered zero until explicitly defined, implying that instances with no keywords from this information group are, in fact, valid. Duplicating or conflicting information is a failure to comply with the standard. Consequently, two coefficients written to the same position in a matrix (or to transposed positions in a symmetric matrix) is an error.

The affine expressions of the objective, g^{obj}, of the scalar constraints, g_i, and of the PSD constraints, G_i, are defined separately. The following notation uses the standard trace inner product for matrices, $\langle X, Y\rangle = \sum_{i,j} X_{ij}Y_{ij}$.

- The affine expression of the objective is defined as

$$g^{obj} = \sum_{j\in\mathcal{J}^{PSD}} \langle F_j^{obj}, X_j\rangle + \sum_{j\in\mathcal{J}} a_j^{obj} x_j + b^{obj},$$

  in terms of the symmetric matrices, F_j^{obj}, and scalars, a_j^{obj} and b^{obj}.

- The affine expressions of the scalar constraints are defined, for $i \in \mathcal{I}$, as

$$g_i = \sum_{j\in\mathcal{J}^{PSD}} \langle F_{ij}, X_j\rangle + \sum_{j\in\mathcal{J}} a_{ij} x_j + b_i,$$

  in terms of the symmetric matrices, F_{ij}, and scalars, a_{ij} and b_i.

- The affine expressions of the PSD constraints are defined, for $i \in \mathcal{I}^{PSD}$, as

$$G_i = \sum_{j\in\mathcal{J}} x_j H_{ij} + D_i,$$

  in terms of the symmetric matrices, H_{ij} and D_i.

List of cones

The format uses an explicit syntax for symmetric positive semidefinite cones as shown above. For scalar variables and constraints, constructed in vectors, the supported conic domains and their minimum sizes are given as follows.

Table 16.3: Cones available in the CBF format

Name	CBF keyword	Cone family
Free domain	`F`	linear
Positive orthant	`L+`	linear
Negative orthant	`L-`	linear
Fixpoint zero	`L=`	linear
Quadratic cone	`Q`	second-order
Rotated quadratic cone	`QR`	second-order

16.4.4 File Format Keywords

VER

Description: The version of the Conic Benchmark Format used to write the file.
`HEADER`: None
`BODY`: One line formatted as:

```
INT
```

This is the version number.
Must appear exactly once in a file, as the first keyword.

OBJSENSE

Description: Define the objective sense.
`HEADER`: None
`BODY`: One line formatted as:

```
STR
```

having `MIN` indicates minimize, and `MAX` indicates maximize. Capital letters are required.
Must appear exactly once in a file.

PSDVAR

Description: Construct the PSD variables.
`HEADER`: One line formatted as:

```
INT
```

This is the number of PSD variables in the problem.
`BODY`: A list of lines formatted as:

```
INT
```

This indicates the number of rows (equal to the number of columns) in the matrix-valued PSD variable. The number of lines should match the number stated in the header.

VAR

Description: Construct the scalar variables.
`HEADER`: One line formatted as:

```
INT INT
```

This is the number of scalar variables, followed by the number of conic domains they are restricted to.
`BODY`: A list of lines formatted as:

```
STR INT
```

This indicates the cone name (see Table 16.3), and the number of scalar variables restricted to this cone. These numbers should add up to the number of scalar variables stated first in the header. The number of lines should match the second number stated in the header.

INT

Description: Declare integer requirements on a selected subset of scalar variables.
`HEADER`: one line formatted as:

```
INT
```

This is the number of integer scalar variables in the problem.
`BODY`: a list of lines formatted as:

```
INT
```

This indicates the scalar variable index $j \in \mathcal{J}$. The number of lines should match the number stated in the header.
Can only be used after the keyword `VAR`.

PSDCON

Description: Construct the PSD constraints.

`HEADER`: One line formatted as:

```
INT
```

This is the number of PSD constraints in the problem.

`BODY`: A list of lines formatted as:

```
INT
```

This indicates the number of rows (equal to the number of columns) in the matrix-valued affine expression of the PSD constraint. The number of lines should match the number stated in the header.

Can only be used after these keywords: `PSDVAR`, `VAR`.

CON

Description: Construct the scalar constraints.

`HEADER`: One line formatted as:

```
INT INT
```

This is the number of scalar constraints, followed by the number of conic domains they restrict to.

`BODY`: A list of lines formatted as:

```
STR INT
```

This indicates the cone name (see Table 16.3), and the number of affine expressions restricted to this cone. These numbers should add up to the number of scalar constraints stated first in the header. The number of lines should match the second number stated in the header.

Can only be used after these keywords: `PSDVAR`, `VAR`

OBJFCOORD

Description: Input sparse coordinates (quadruplets) to define the symmetric matrices F_j^{obj}, as used in the objective.

`HEADER`: One line formatted as:

```
INT
```

This is the number of coordinates to be specified.

`BODY`: A list of lines formatted as:

```
INT INT INT REAL
```

This indicates the PSD variable index $j \in \mathcal{J}^{PSD}$, the row index, the column index and the coefficient value. The number of lines should match the number stated in the header.

OBJACOORD

Description: Input sparse coordinates (pairs) to define the scalars, a_j^{obj}, as used in the objective.

`HEADER`: One line formatted as:

```
INT
```

This is the number of coordinates to be specified.

`BODY`: A list of lines formatted as:

```
INT REAL
```

This indicates the scalar variable index $j \in \mathcal{J}$ and the coefficient value. The number of lines should match the number stated in the header.

OBJBCOORD

Description: Input the scalar, b^{obj}, as used in the objective.
`HEADER`: None.
`BODY`: One line formatted as:

```
REAL
```

This indicates the coefficient value.

FCOORD

Description: Input sparse coordinates (quintuplets) to define the symmetric matrices, F_{ij}, as used in the scalar constraints.
`HEADER`: One line formatted as:

```
INT
```

This is the number of coordinates to be specified.
`BODY`: A list of lines formatted as:

```
INT INT INT INT REAL
```

This indicates the scalar constraint index $i \in \mathcal{I}$, the PSD variable index $j \in \mathcal{J}^{PSD}$, the row index, the column index and the coefficient value. The number of lines should match the number stated in the header.

ACOORD

Description: Input sparse coordinates (triplets) to define the scalars, a_{ij}, as used in the scalar constraints.
`HEADER`: One line formatted as:

```
INT
```

This is the number of coordinates to be specified.
`BODY`: A list of lines formatted as:

```
INT INT REAL
```

This indicates the scalar constraint index $i \in \mathcal{I}$, the scalar variable index $j \in \mathcal{J}$ and the coefficient value. The number of lines should match the number stated in the header.

BCOORD

Description: Input sparse coordinates (pairs) to define the scalars, b_i, as used in the scalar constraints.
`HEADER`: One line formatted as:

```
INT
```

This is the number of coordinates to be specified.
`BODY`: A list of lines formatted as:

```
INT REAL
```

This indicates the scalar constraint index $i \in \mathcal{I}$ and the coefficient value. The number of lines should match the number stated in the header.

HCOORD

Description: Input sparse coordinates (quintuplets) to define the symmetric matrices, H_{ij}, as used in the PSD constraints.
`HEADER`: One line formatted as:

```
INT
```

This is the number of coordinates to be specified.

BODY: A list of lines formatted as

```
INT INT INT INT REAL
```

This indicates the PSD constraint index $i \in \mathcal{I}^{PSD}$, the scalar variable index $j \in \mathcal{J}$, the row index, the column index and the coefficient value. The number of lines should match the number stated in the header.

DCOORD

Description: Input sparse coordinates (quadruplets) to define the symmetric matrices, D_i, as used in the PSD constraints.

HEADER: One line formatted as

```
INT
```

This is the number of coordinates to be specified.

BODY: A list of lines formatted as:

```
INT INT INT REAL
```

This indicates the PSD constraint index $i \in \mathcal{I}^{PSD}$, the row index, the column index and the coefficient value. The number of lines should match the number stated in the header.

CHANGE

Start of a new instance specification based on changes to the previous. Can be interpreted as the end of file when the hotstart-sequence is unsupported or undesired.

BODY: None

Header: None

16.4.5 CBF Format Examples

Minimal Working Example

The conic optimization problem (16.11) , has three variables in a quadratic cone - first one is integer - and an affine expression in domain 0 (equality constraint).

$$\begin{array}{ll} \text{minimize} & 5.1\,x_0 \\ \text{subject to} & 6.2\,x_1 + 7.3\,x_2 - 8.4 \in \{0\} \\ & x \in \mathcal{Q}^3,\ x_0 \in \mathbb{Z}. \end{array} \tag{16.11}$$

Its formulation in the Conic Benchmark Format begins with the version of the CBF format used, to safeguard against later revisions.

```
VER
1
```

Next follows the problem structure, consisting of the objective sense, the number and domain of variables, the indices of integer variables, and the number and domain of scalar-valued affine expressions (i.e., the equality constraint).

```
OBJSENSE
MIN

VAR
3 1
Q 3

INT
1
0
```

(continues on next page)

(continued from previous page)

```
CON
1 1
L= 1
```

Finally follows the problem data, consisting of the coefficients of the objective, the coefficients of the constraints, and the constant terms of the constraints. All data is specified on a sparse coordinate form.

```
OBJACOORD
1
0 5.1

ACOORD
2
0 1 6.2
0 2 7.3

BCOORD
1
0 -8.4
```

This concludes the example.

Mixing Linear, Second-order and Semidefinite Cones

The conic optimization problem (16.12), has a semidefinite cone, a quadratic cone over unordered subindices, and two equality constraints.

$$\begin{array}{lccccc}
\text{minimize} & \left\langle \begin{bmatrix} 2 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 2 \end{bmatrix}, X_1 \right\rangle + x_1 & & \\
\text{subject to} & \left\langle \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, X_1 \right\rangle + x_1 & = & 1.0\,, \\
& \left\langle \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, X_1 \right\rangle + x_0 + x_2 & = & 0.5\,, \\
& x_1 \geq \sqrt{x_0^2 + x_2^2}\,, & & \\
& X_1 \succeq \mathbf{0}\,. & &
\end{array} \tag{16.12}$$

The equality constraints are easily rewritten to the conic form, $(g_0, g_1) \in \{0\}^2$, by moving constants such that the right-hand-side becomes zero. The quadratic cone does not fit under the `VAR` keyword in this variable permutation. Instead, it takes a scalar constraint $(g_2, g_3, g_4) = (x_1, x_0, x_2) \in \mathcal{Q}^3$, with scalar variables constructed as $(x_0, x_1, x_2) \in \mathbb{R}^3$. Its formulation in the CBF format is reported in the following list

```
# File written using this version of the Conic Benchmark Format:
#     | Version 1.
VER
1

# The sense of the objective is:
#     | Minimize.
OBJSENSE
MIN

# One PSD variable of this size:
#     | Three times three.
PSDVAR
1
3
```

(continues on next page)

(continued from previous page)

```
# Three scalar variables in this one conic domain:
#     | Three are free.
VAR
3 1
F 3

# Five scalar constraints with affine expressions in two conic domains:
#     | Two are fixed to zero.
#     | Three are in conic quadratic domain.
CON
5 2
L= 2
Q 3

# Five coordinates in F^{obj}_j coefficients:
#     | F^{obj}[0][0,0] = 2.0
#     | F^{obj}[0][1,0] = 1.0
#     | and more...
OBJFCOORD
5
0 0 0 2.0
0 1 0 1.0
0 1 1 2.0
0 2 1 1.0
0 2 2 2.0

# One coordinate in a^{obj}_j coefficients:
#     | a^{obj}[1] = 1.0
OBJACOORD
1
1 1.0

# Nine coordinates in F_ij coefficients:
#     | F[0,0][0,0] = 1.0
#     | F[0,0][1,1] = 1.0
#     | and more...
FCOORD
9
0 0 0 0 1.0
0 0 1 1 1.0
0 0 2 2 1.0
1 0 0 0 1.0
1 0 1 0 1.0
1 0 2 0 1.0
1 0 1 1 1.0
1 0 2 1 1.0
1 0 2 2 1.0

# Six coordinates in a_ij coefficients:
#     | a[0,1] = 1.0
#     | a[1,0] = 1.0
#     | and more...
ACOORD
6
0 1 1.0
1 0 1.0
1 2 1.0
2 1 1.0
3 0 1.0
4 2 1.0
```

(continues on next page)

(continued from previous page)

```
# Two coordinates in b_i coefficients:
#     | b[0] = -1.0
#     | b[1] = -0.5
BCOORD
2
0 -1.0
1 -0.5
```

Mixing Semidefinite Variables and Linear Matrix Inequalities

The standard forms in semidefinite optimization are usually based either on semidefinite variables or linear matrix inequalities. In the CBF format, both forms are supported and can even be mixed as shown in.

$$
\begin{array}{lllr}
\text{minimize} & \left\langle \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, X_1 \right\rangle + x_1 + x_2 + 1 & & \\
\text{subject to} & \left\langle \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, X_1 \right\rangle - x_1 - x_2 & \geq \; 0.0\,, & \\
 & x_1 \begin{bmatrix} 0 & 1 \\ 1 & 3 \end{bmatrix} + x_2 \begin{bmatrix} 3 & 1 \\ 1 & 0 \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \succeq \; \mathbf{0}\,, & \\
 & X_1 \succeq \mathbf{0}\,. & &
\end{array} \qquad (16.13)
$$

Its formulation in the CBF format is written in what follows

```
# File written using this version of the Conic Benchmark Format:
#     | Version 1.
VER
1

# The sense of the objective is:
#     | Minimize.
OBJSENSE
MIN

# One PSD variable of this size:
#     | Two times two.
PSDVAR
1
2

# Two scalar variables in this one conic domain:
#     | Two are free.
VAR
2 1
F 2

# One PSD constraint of this size:
#     | Two times two.
PSDCON
1
2

# One scalar constraint with an affine expression in this one conic domain:
#     | One is greater than or equal to zero.
CON
1 1
L+ 1

# Two coordinates in F^{obj}_j coefficients:
```

(continues on next page)

(continued from previous page)

```
#     | F^{obj}[0][0,0] = 1.0
#     | F^{obj}[0][1,1] = 1.0
OBJFCOORD
2
0 0 0 1.0
0 1 1 1.0

# Two coordinates in a^{obj}_j coefficients:
#     | a^{obj}[0] = 1.0
#     | a^{obj}[1] = 1.0
OBJACOORD
2
0 1.0
1 1.0

# One coordinate in b^{obj} coefficient:
#     | b^{obj} = 1.0
OBJBCOORD
1.0

# One coordinate in F_ij coefficients:
#     | F[0,0][1,0] = 1.0
FCOORD
1
0 0 1 0 1.0

# Two coordinates in a_ij coefficients:
#     | a[0,0] = -1.0
#     | a[0,1] = -1.0
ACOORD
2
0 0 -1.0
0 1 -1.0

# Four coordinates in H_ij coefficients:
#     | H[0,0][1,0] = 1.0
#     | H[0,0][1,1] = 3.0
#     | and more...
HCOORD
4
0 0 1 0 1.0
0 0 1 1 3.0
0 1 0 0 3.0
0 1 1 0 1.0

# Two coordinates in D_i coefficients:
#     | D[0][0,0] = -1.0
#     | D[0][1,1] = -1.0
DCOORD
2
0 0 0 -1.0
0 1 1 -1.0
```

Optimization Over a Sequence of Objectives

The linear optimization problem (16.14), is defined for a sequence of objectives such that hotstarting from one to the next might be advantages.

$$
\begin{array}{lrcl}
\text{maximize}_k & g_k^{obj} & & \\
\text{subject to} & 50\,x_0 + 31 & \leq & 250\,, \\
 & 3\,x_0 - 2x_1 & \geq & -4\,, \\
 & x \in \mathbb{R}_+^2, & &
\end{array}
\tag{16.14}
$$

given,

1. $g_0^{obj} = x_0 + 0.64x_1$.
2. $g_1^{obj} = 1.11x_0 + 0.76x_1$.
3. $g_2^{obj} = 1.11x_0 + 0.85x_1$.

Its formulation in the CBF format is reported in Listing 16.5.

Listing 16.5: Problem (16.14) in CBF format.

```
# File written using this version of the Conic Benchmark Format:
#     | Version 1.
VER
1

# The sense of the objective is:
#     | Maximize.
OBJSENSE
MAX

# Two scalar variables in this one conic domain:
#     | Two are nonnegative.
VAR
2 1
L+ 2

# Two scalar constraints with affine expressions in these two conic domains:
#     | One is in the nonpositive domain.
#     | One is in the nonnegative domain.
CON
2 2
L- 1
L+ 1

# Two coordinates in a^{obj}_j coefficients:
#     | a^{obj}[0] = 1.0
#     | a^{obj}[1] = 0.64
OBJACOORD
2
0 1.0
1 0.64

# Four coordinates in a_ij coefficients:
#     | a[0,0] = 50.0
#     | a[1,0] = 3.0
#     | and more...
ACOORD
4
0 0 50.0
1 0 3.0
0 1 31.0
1 1 -2.0

# Two coordinates in b_i coefficients:
#     | b[0] = -250.0
#     | b[1] = 4.0
BCOORD
2
0 -250.0
1 4.0
```

(continues on next page)

(continued from previous page)

```
# New problem instance defined in terms of changes.
CHANGE

# Two coordinate changes in a^{obj}_j coefficients. Now it is:
#     | a^{obj}[0] = 1.11
#     | a^{obj}[1] = 0.76
OBJACOORD
2
0 1.11
1 0.76

# New problem instance defined in terms of changes.
CHANGE

# One coordinate change in a^{obj}_j coefficients. Now it is:
#     | a^{obj}[0] = 1.11
#     | a^{obj}[1] = 0.85
OBJACOORD
1
1 0.85
```

16.5 The PTF Format

The PTF format is a new human-readable, natural text format. Its features and structure are similar to the *OPF* format, with the difference that the PTF format **does** support semidefinite terms.

16.5.1 The overall format

The format is indentation based, where each section is started by a head line and followed by a section body with deeper indentation that the head line. For example:

```
Header line
   Body line 1
   Body line 1
   Body line 1
```

Section can also be nested:

```
Header line A
   Body line in A
   Header line A.1
      Body line in A.1
      Body line in A.1
   Body line in A
```

The indentation of blank lines is ignored, so a subsection can contain a blank line with no indentation. The character # defines a line comment and anything between the # character and the end of the line is ignored.

In a PTF file, the first section must be a `Task` section. The order of the remaining section is arbitrary, and sections may occur multiple times or not at all.

MOSEK will ignore any top-level section it does not recognize.

Names

In the description of the format we use following definitions for name strings:

```
NAME: PLAIN_NAME | QUOTED_NAME
PLAIN_NAME: [a-zA-Z_] [a-zA-Z0-9_-.!|]
QUOTED_NAME: "'" ( [^'\\\r\n] | "\\" ( [\\rn] | "x" [0-9a-fA-F] [0-9a-fA-F] ) )* "'"
```

Expressions

An expression is a sum of terms. A term is either a linear term (a coefficient and a variable name, where the coefficient can be left out if it is 1.0), or a matrix inner product.

An expression:

```
EXPR: EMPTY | [+-]? TERM ( [+-] TERM )*
TERM: LINEAR_TERM | MATRIX_TERM
```

A linear term

```
LINEAR_TERM: FLOAT? NAME
```

A matrix term

```
MATRIX_TERM: "<" FLOAT? NAME ( [+-] FLOAT? NAME)* ";" NAME ">"
```

Here the right-hand name is the name of a (semidefinite) matrix variable, and the left-hand side is a sum of symmetric matrixes. The actual matrixes are defined in a separate section.

Expressions can span multiple lines by giving subsequent lines a deeper indentation.

For example following two section are equivalent:

```
# Everything on one line:
x1 + x2 + x3 + x4

# Split into multiple lines:
x1
  + x2
  + x3
  + x4
```

16.5.2 `Task` section

The first section of the file must be a `Task`. The text in this section is not used and may contain comments, or meta-information from the writer or about the content.

Format:

```
Task NAME
   Anything goes here...
```

`NAME` is a the task name.

16.5.3 `Objective` section

The `Objective` section defines the objective name, sense and function. The format:

```
"Objective" NAME?
   ( "Minimize" | "Maximize" ) EXPR
```

For example:

```
Objective 'obj'
   Minimize x1 + 0.2 x2 + < M1 ; X1 >
```

16.5.4 `Constraints` section

The constraints section defines a series of constraints. A constraint defines a term $A \cdot x + b \in K$. For linear constraints `A` is just one row, while for conic constraints it can be multiple rows. If a constraint spans multiple rows these can either be written inline separated by semi-colons, or each expression in a separete sub-section.

Simple linear constraints:

```
"Constraints"
  NAME? "[" [-+] (FLOAT | "Inf") (";" [-+] (FLOAT | "Inf") )? "]" EXPR
```

If the brackets contain two values, they are used as upper and lower bounds. It they contain one value the constraint is an equality.

For example:

```
Constraints
  'c1' [0;10] x1 + x2 + x3
  [0] x1 + x2 + x3
```

Constraint blocks put the expression either in a subsection or inline. The cone type (domain) is written in the brackets, and **MOSEK** currently supports following types:

- `SOC(N)` Second order cone of dimension `N`
- `RSOC(N)` Rotated second order cone of dimension `N`
- `PSD(N)` Symmetric positive semidefinite cone of dimension `N`. This contains `N*(N+1)/2` elements.
- `PEXP` Primal exponential cone of dimension 3
- `DEXP` Dual exponential cone of dimension 3
- `PPOW(N,P)` Primal power cone of dimension `N` with parameter `P`
- `DPOW(N,P)` Dual power cone of dimension `N` with parameter `P`
- `ZERO(N)` The zero-cone of dimension `N`.

```
"Constraints"
  NAME? "[" DOMAIN "]" EXPR_LIST
```

For example:

```
Constraints
   'K1' [SOC(3)] x1 + x2 ; x2 + x3 ; x3 + x1
   'K2' [RSOC(3)]
      x1 + x2
      x2 + x3
      x3 + x1
```

16.5.5 `Variables` section

Any variable used in an expression must be defined in a variable section. The variable section defines each variable domain.

```
"Variables"
   NAME "[" [-+] (FLOAT | "Inf") (";" [-+] (FLOAT | "Inf") )? "]"
   NAME "[" DOMAIN "]" NAMES

   For example, a linear variable
```

```
Variables
   x1 [0;Inf]
```

As with constraints, members of a conic domain can be listed either inline or in a subsection:

```
Variables
   k1 [SOC(3)] x1 ; x2 ; x3
   k2 [RSOC(3)]
      x1
      x2
      x3
```

16.5.6 Integer section

This section contains a list of variables that are integral. For example:

```
Integer
   x1 x2 x3
```

16.5.7 SymmetricMatrixes section

This section defines the symmetric matrixes used for matrix coefficients in matrix inner product terms. The section lists named matrixes, each with a size and a number of non-zeros. Only non-zeros in the lower triangular part should be defined.

```
"SymmetricMatrixes"
   NAME "SYMMAT" "(" INT ")"  ( "(" INT "," INT "," FLOAT ")" )*
   ...
```

For example:

```
SymmetricMatrixes
   M1 SYMMAT(3) (0,0,1.0) (1,1,2.0) (2,1,0.5)
   M2 SYMMAT(3)
      (0,0,1.0)
      (1,1,2.0)
      (2,1,0.5)
```

16.5.8 Solutions section

Each subsection defines a solution. A solution defines for each constraint and for each variable exactly one primal value and either one (for conic domains) or two (for linear domains) dual values. The values follow the same logic as in the **MOSEK** C API. A primal and a dual solution status defines the meaning of the values primal and dual (solution, certificate, unknown, etc.)

The format is this:

```
"Solutions"
   "Solution" WHICHSOL
      "ProblemStatus" PROSTA PROSTA?
   "SolutionStatus" SOLSTA SOLSTA?
   "Objective" FLOAT FLOAT
   "Variables"
      # Linear variable status: level, slx, sux
      NAME "[" STATUS "]" FLOAT (FLOAT FLOAT)?
      # Conic variable status: level, snx
      NAME
         "[" STATUS "]" FLOAT FLOAT?
         ...
   "Constraints"
      # Linear variable status: level, slx, sux
      NAME "[" STATUS "]" FLOAT (FLOAT FLOAT)?
      # Conic variable status: level, snx
      NAME
         "[" STATUS "]" FLOAT FLOAT?
         ...
```

Following values for WHICHSOL are supported:

- interior Interior solution, the result of an interior-point solver.
- basic Basic solution, as produced by a simplex solver.
- integer Integer solution, the solution to a mixed-integer problem. This does not define a dual solution.

Following values for PROSTA are supported:

- `unknown` The problem status is unknown
- `feasible` The problem has been proven feasible
- `infeasible` The problem has been proven infeasible
- `illposed` The problem has been proved to be ill posed
- `infeasible_or_unbounded` The problem is infeasible or unbounded

Following values for `SOLSTA` are supported:

- `unknown` The solution status is unknown
- `feasible` The solution is feasible
- `optimal` The solution is optimal
- `infeas_cert` The solution is a certificate of infeasibility
- `illposed_cert` The solution is a certificate of illposedness

Following values for `STATUS` are supported:

- `unknown` The value is unknown
- `super_basic` The value is super basic
- `at_lower` The value is basic and at its lower bound
- `at_upper` The value is basic and at its upper bound
- `fixed` The value is basic fixed
- `infinite` The value is at infinity

16.6 The Task Format

The Task format is **MOSEK**'s native binary format. It contains a complete image of a **MOSEK** task, i.e.

- Problem data: Linear, conic, semidefinite and quadratic data
- Problem item names: Variable names, constraints names, cone names etc.
- Parameter settings
- Solutions

There are a few things to be aware of:

- Status of a solution read from a file will *always* be unknown.
- Parameter settings in a task file *always override* any parameters set on the command line or in a parameter file.

The format is based on the *TAR* (USTar) file format. This means that the individual pieces of data in a `.task` file can be examined by unpacking it as a *TAR* file. Please note that the inverse may not work: Creating a file using *TAR* will most probably not create a valid **MOSEK** Task file since the order of the entries is important.

16.7 The JSON Format

MOSEK provides the possibility to read/write problems in valid JSON format.

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.

The official JSON website http://www.json.org provides plenty of information along with the format definition.

MOSEK defines two JSON-like formats:

- `jtask`
- `jsol`

Despite being text-based human-readable formats, *jtask* and *jsol* files will include no indentation and no new-lines, in order to keep the files as compact as possible. We therefore strongly advise to use JSON viewer tools to inspect *jtask* and *jsol* files.

16.7.1 *jtask* format

It stores a problem instance. The *jtask* format contains the same information as a *task format*. Even though a *jtask* file is human-readable, we do not recommend users to create it by hand, but to rely on **MOSEK**.

16.7.2 *jsol* format

It stores a problem solution. The *jsol* format contains all solutions and information items.

You can write a *jsol* file using `Task.writejsonsol`. You **can not** read a *jsol* file into **MOSEK**.

16.7.3 A *jtask* example

In Listing 16.6 we present a file in the *jtask* format that corresponds to the sample problem from `lo1.lp`. The listing has been formatted for readability.

Listing 16.6: A formatted *jtask* file for the `lo1.lp` example.

```
{
    "$schema":"http://mosek.com/json/schema#",
    "Task/INFO":{
        "taskname":"lo1",
        "numvar":4,
        "numcon":3,
        "numcone":0,
        "numbarvar":0,
        "numanz":9,
        "numsymmat":0,
        "mosekver":[
            8,
            0,
            0,
            9
        ]
    },
    "Task/data":{
        "var":{
            "name":[
                "x1",
                "x2",
```

(continues on next page)

(continued from previous page)

```
                "x3",
                "x4"
            ],
            "bk":[
                "lo",
                "ra",
                "lo",
                "lo"
            ],
            "bl":[
                0.0,
                0.0,
                0.0,
                0.0
            ],
            "bu":[
                1e+30,
                1e+1,
                1e+30,
                1e+30
            ],
            "type":[
                "cont",
                "cont",
                "cont",
                "cont"
            ]
        },
        "con":{
            "name":[
                "c1",
                "c2",
                "c3"
            ],
            "bk":[
                "fx",
                "lo",
                "up"
            ],
            "bl":[
                3e+1,
                1.5e+1,
                    -1e+30
            ],
            "bu":[
                3e+1,
                1e+30,
                2.5e+1
            ]
        },
        "objective":{
            "sense":"max",
            "name":"obj",
            "c":{
                "subj":[
                    0,
                    1,
                    2,
                    3
                ],
                "val":[
```

(continues on next page)

(continued from previous page)


```
                3e+0,
                1e+0,
                5e+0,
                1e+0
            ]
        },
        "cfix":0.0
    },
    "A":{
        "subi":[
            0,
            0,
            0,
            1,
            1,
            1,
            1,
            2,
            2
        ],
        "subj":[
            0,
            1,
            2,
            0,
            1,
            2,
            3,
            1,
            3
        ],
        "val":[
            3e+0,
            1e+0,
            2e+0,
            2e+0,
            1e+0,
            3e+0,
            1e+0,
            2e+0,
            3e+0
        ]
    }
},
"Task/parameters":{
    "iparam":{
        "ANA_SOL_BASIS":"ON",
        "ANA_SOL_PRINT_VIOLATED":"OFF",
        "AUTO_SORT_A_BEFORE_OPT":"OFF",
        "AUTO_UPDATE_SOL_INFO":"OFF",
        "BASIS_SOLVE_USE_PLUS_ONE":"OFF",
        "BI_CLEAN_OPTIMIZER":"OPTIMIZER_FREE",
        "BI_IGNORE_MAX_ITER":"OFF",
        "BI_IGNORE_NUM_ERROR":"OFF",
        "BI_MAX_ITERATIONS":1000000,
        "CACHE_LICENSE":"ON",
        "CHECK_CONVEXITY":"CHECK_CONVEXITY_FULL",
        "COMPRESS_STATFILE":"ON",
        "CONCURRENT_NUM_OPTIMIZERS":2,
        "CONCURRENT_PRIORITY_DUAL_SIMPLEX":2,
        "CONCURRENT_PRIORITY_FREE_SIMPLEX":3,
```


(continues on next page)

(continued from previous page)

```
"CONCURRENT_PRIORITY_INTPNT":4,
"CONCURRENT_PRIORITY_PRIMAL_SIMPLEX":1,
"FEASREPAIR_OPTIMIZE":"FEASREPAIR_OPTIMIZE_NONE",
"INFEAS_GENERIC_NAMES":"OFF",
"INFEAS_PREFER_PRIMAL":"ON",
"INFEAS_REPORT_AUTO":"OFF",
"INFEAS_REPORT_LEVEL":1,
"INTPNT_BASIS":"BI_ALWAYS",
"INTPNT_DIFF_STEP":"ON",
"INTPNT_FACTOR_DEBUG_LVL":0,
"INTPNT_FACTOR_METHOD":0,
"INTPNT_HOTSTART":"INTPNT_HOTSTART_NONE",
"INTPNT_MAX_ITERATIONS":400,
"INTPNT_MAX_NUM_COR":-1,
"INTPNT_MAX_NUM_REFINEMENT_STEPS":-1,
"INTPNT_OFF_COL_TRH":40,
"INTPNT_ORDER_METHOD":"ORDER_METHOD_FREE",
"INTPNT_REGULARIZATION_USE":"ON",
"INTPNT_SCALING":"SCALING_FREE",
"INTPNT_SOLVE_FORM":"SOLVE_FREE",
"INTPNT_STARTING_POINT":"STARTING_POINT_FREE",
"LIC_TRH_EXPIRY_WRN":7,
"LICENSE_DEBUG":"OFF",
"LICENSE_PAUSE_TIME":0,
"LICENSE_SUPPRESS_EXPIRE_WRNS":"OFF",
"LICENSE_WAIT":"OFF",
"LOG":10,
"LOG_ANA_PRO":1,
"LOG_BI":4,
"LOG_BI_FREQ":2500,
"LOG_CHECK_CONVEXITY":0,
"LOG_CONCURRENT":1,
"LOG_CUT_SECOND_OPT":1,
"LOG_EXPAND":0,
"LOG_FACTOR":1,
"LOG_FEAS_REPAIR":1,
"LOG_FILE":1,
"LOG_HEAD":1,
"LOG_INFEAS_ANA":1,
"LOG_INTPNT":4,
"LOG_MIO":4,
"LOG_MIO_FREQ":1000,
"LOG_OPTIMIZER":1,
"LOG_ORDER":1,
"LOG_PRESOLVE":1,
"LOG_RESPONSE":0,
"LOG_SENSITIVITY":1,
"LOG_SENSITIVITY_OPT":0,
"LOG_SIM":4,
"LOG_SIM_FREQ":1000,
"LOG_SIM_MINOR":1,
"LOG_STORAGE":1,
"MAX_NUM_WARNINGS":10,
"MIO_BRANCH_DIR":"BRANCH_DIR_FREE",
"MIO_CONSTRUCT_SOL":"OFF",
"MIO_CUT_CLIQUE":"ON",
"MIO_CUT_CMIR":"ON",
"MIO_CUT_GMI":"ON",
"MIO_CUT_KNAPSACK_COVER":"OFF",
"MIO_HEURISTIC_LEVEL":-1,
"MIO_MAX_NUM_BRANCHES":-1,
```

(continues on next page)

(continued from previous page)

```
            "MIO_MAX_NUM_RELAXS":-1,
            "MIO_MAX_NUM_SOLUTIONS":-1,
            "MIO_MODE":"MIO_MODE_SATISFIED",
            "MIO_MT_USER_CB":"ON",
            "MIO_NODE_OPTIMIZER":"OPTIMIZER_FREE",
            "MIO_NODE_SELECTION":"MIO_NODE_SELECTION_FREE",
            "MIO_PERSPECTIVE_REFORMULATE":"ON",
            "MIO_PROBING_LEVEL":-1,
            "MIO_RINS_MAX_NODES":-1,
            "MIO_ROOT_OPTIMIZER":"OPTIMIZER_FREE",
            "MIO_ROOT_REPEAT_PRESOLVE_LEVEL":-1,
            "MT_SPINCOUNT":0,
            "NUM_THREADS":0,
            "OPF_MAX_TERMS_PER_LINE":5,
            "OPF_WRITE_HEADER":"ON",
            "OPF_WRITE_HINTS":"ON",
            "OPF_WRITE_PARAMETERS":"OFF",
            "OPF_WRITE_PROBLEM":"ON",
            "OPF_WRITE_SOL_BAS":"ON",
            "OPF_WRITE_SOL_ITG":"ON",
            "OPF_WRITE_SOL_ITR":"ON",
            "OPF_WRITE_SOLUTIONS":"OFF",
            "OPTIMIZER":"OPTIMIZER_FREE",
            "PARAM_READ_CASE_NAME":"ON",
            "PARAM_READ_IGN_ERROR":"OFF",
            "PRESOLVE_ELIMINATOR_MAX_FILL":-1,
            "PRESOLVE_ELIMINATOR_MAX_NUM_TRIES":-1,
            "PRESOLVE_LEVEL":-1,
            "PRESOLVE_LINDEP_ABS_WORK_TRH":100,
            "PRESOLVE_LINDEP_REL_WORK_TRH":100,
            "PRESOLVE_LINDEP_USE":"ON",
            "PRESOLVE_MAX_NUM_REDUCTIONS":-1,
            "PRESOLVE_USE":"PRESOLVE_MODE_FREE",
            "PRIMAL_REPAIR_OPTIMIZER":"OPTIMIZER_FREE",
            "QO_SEPARABLE_REFORMULATION":"OFF",
            "READ_DATA_COMPRESSED":"COMPRESS_FREE",
            "READ_DATA_FORMAT":"DATA_FORMAT_EXTENSION",
            "READ_DEBUG":"OFF",
            "READ_KEEP_FREE_CON":"OFF",
            "READ_LP_DROP_NEW_VARS_IN_BOU":"OFF",
            "READ_LP_QUOTED_NAMES":"ON",
            "READ_MPS_FORMAT":"MPS_FORMAT_FREE",
            "READ_MPS_WIDTH":1024,
            "READ_TASK_IGNORE_PARAM":"OFF",
            "SENSITIVITY_ALL":"OFF",
            "SENSITIVITY_OPTIMIZER":"OPTIMIZER_FREE_SIMPLEX",
            "SENSITIVITY_TYPE":"SENSITIVITY_TYPE_BASIS",
            "SIM_BASIS_FACTOR_USE":"ON",
            "SIM_DEGEN":"SIM_DEGEN_FREE",
            "SIM_DUAL_CRASH":90,
            "SIM_DUAL_PHASEONE_METHOD":0,
            "SIM_DUAL_RESTRICT_SELECTION":50,
            "SIM_DUAL_SELECTION":"SIM_SELECTION_FREE",
            "SIM_EXPLOIT_DUPVEC":"SIM_EXPLOIT_DUPVEC_OFF",
            "SIM_HOTSTART":"SIM_HOTSTART_FREE",
            "SIM_HOTSTART_LU":"ON",
            "SIM_INTEGER":0,
            "SIM_MAX_ITERATIONS":10000000,
            "SIM_MAX_NUM_SETBACKS":250,
            "SIM_NON_SINGULAR":"ON",
            "SIM_PRIMAL_CRASH":90,
```

(continues on next page)

(continued from previous page)


```
        "SIM_PRIMAL_PHASEONE_METHOD":0,
        "SIM_PRIMAL_RESTRICT_SELECTION":50,
        "SIM_PRIMAL_SELECTION":"SIM_SELECTION_FREE",
        "SIM_REFACTOR_FREQ":0,
        "SIM_REFORMULATION":"SIM_REFORMULATION_OFF",
        "SIM_SAVE_LU":"OFF",
        "SIM_SCALING":"SCALING_FREE",
        "SIM_SCALING_METHOD":"SCALING_METHOD_POW2",
        "SIM_SOLVE_FORM":"SOLVE_FREE",
        "SIM_STABILITY_PRIORITY":50,
        "SIM_SWITCH_OPTIMIZER":"OFF",
        "SOL_FILTER_KEEP_BASIC":"OFF",
        "SOL_FILTER_KEEP_RANGED":"OFF",
        "SOL_READ_NAME_WIDTH":-1,
        "SOL_READ_WIDTH":1024,
        "SOLUTION_CALLBACK":"OFF",
        "TIMING_LEVEL":1,
        "WRITE_BAS_CONSTRAINTS":"ON",
        "WRITE_BAS_HEAD":"ON",
        "WRITE_BAS_VARIABLES":"ON",
        "WRITE_DATA_COMPRESSED":0,
        "WRITE_DATA_FORMAT":"DATA_FORMAT_EXTENSION",
        "WRITE_DATA_PARAM":"OFF",
        "WRITE_FREE_CON":"OFF",
        "WRITE_GENERIC_NAMES":"OFF",
        "WRITE_GENERIC_NAMES_IO":1,
        "WRITE_IGNORE_INCOMPATIBLE_CONIC_ITEMS":"OFF",
        "WRITE_IGNORE_INCOMPATIBLE_ITEMS":"OFF",
        "WRITE_IGNORE_INCOMPATIBLE_NL_ITEMS":"OFF",
        "WRITE_IGNORE_INCOMPATIBLE_PSD_ITEMS":"OFF",
        "WRITE_INT_CONSTRAINTS":"ON",
        "WRITE_INT_HEAD":"ON",
        "WRITE_INT_VARIABLES":"ON",
        "WRITE_LP_FULL_OBJ":"ON",
        "WRITE_LP_LINE_WIDTH":80,
        "WRITE_LP_QUOTED_NAMES":"ON",
        "WRITE_LP_STRICT_FORMAT":"OFF",
        "WRITE_LP_TERMS_PER_LINE":10,
        "WRITE_MPS_FORMAT":"MPS_FORMAT_FREE",
        "WRITE_MPS_INT":"ON",
        "WRITE_PRECISION":15,
        "WRITE_SOL_BARVARIABLES":"ON",
        "WRITE_SOL_CONSTRAINTS":"ON",
        "WRITE_SOL_HEAD":"ON",
        "WRITE_SOL_IGNORE_INVALID_NAMES":"OFF",
        "WRITE_SOL_VARIABLES":"ON",
        "WRITE_TASK_INC_SOL":"ON",
        "WRITE_XML_MODE":"WRITE_XML_MODE_ROW"
    },
    "dparam":{
        "ANA_SOL_INFEAS_TOL":1e-6,
        "BASIS_REL_TOL_S":1e-12,
        "BASIS_TOL_S":1e-6,
        "BASIS_TOL_X":1e-6,
        "CHECK_CONVEXITY_REL_TOL":1e-10,
        "DATA_TOL_AIJ":1e-12,
        "DATA_TOL_AIJ_HUGE":1e+20,
        "DATA_TOL_AIJ_LARGE":1e+10,
        "DATA_TOL_BOUND_INF":1e+16,
        "DATA_TOL_BOUND_WRN":1e+8,
        "DATA_TOL_C_HUGE":1e+16,
```


(continues on next page)

(continued from previous page)


```
        "DATA_TOL_CJ_LARGE":1e+8,
        "DATA_TOL_QIJ":1e-16,
        "DATA_TOL_X":1e-8,
        "FEASREPAIR_TOL":1e-10,
        "INTPNT_CO_TOL_DFEAS":1e-8,
        "INTPNT_CO_TOL_INFEAS":1e-10,
        "INTPNT_CO_TOL_MU_RED":1e-8,
        "INTPNT_CO_TOL_NEAR_REL":1e+3,
        "INTPNT_CO_TOL_PFEAS":1e-8,
        "INTPNT_CO_TOL_REL_GAP":1e-7,
        "INTPNT_NL_MERIT_BAL":1e-4,
        "INTPNT_NL_TOL_DFEAS":1e-8,
        "INTPNT_NL_TOL_MU_RED":1e-12,
        "INTPNT_NL_TOL_NEAR_REL":1e+3,
        "INTPNT_NL_TOL_PFEAS":1e-8,
        "INTPNT_NL_TOL_REL_GAP":1e-6,
        "INTPNT_NL_TOL_REL_STEP":9.95e-1,
        "INTPNT_QO_TOL_DFEAS":1e-8,
        "INTPNT_QO_TOL_INFEAS":1e-10,
        "INTPNT_QO_TOL_MU_RED":1e-8,
        "INTPNT_QO_TOL_NEAR_REL":1e+3,
        "INTPNT_QO_TOL_PFEAS":1e-8,
        "INTPNT_QO_TOL_REL_GAP":1e-8,
        "INTPNT_TOL_DFEAS":1e-8,
        "INTPNT_TOL_DSAFE":1e+0,
        "INTPNT_TOL_INFEAS":1e-10,
        "INTPNT_TOL_MU_RED":1e-16,
        "INTPNT_TOL_PATH":1e-8,
        "INTPNT_TOL_PFEAS":1e-8,
        "INTPNT_TOL_PSAFE":1e+0,
        "INTPNT_TOL_REL_GAP":1e-8,
        "INTPNT_TOL_REL_STEP":9.999e-1,
        "INTPNT_TOL_STEP_SIZE":1e-6,
        "LOWER_OBJ_CUT":-1e+30,
        "LOWER_OBJ_CUT_FINITE_TRH":-5e+29,
        "MIO_DISABLE_TERM_TIME":-1e+0,
        "MIO_MAX_TIME":-1e+0,
        "MIO_MAX_TIME_APRX_OPT":6e+1,
        "MIO_NEAR_TOL_ABS_GAP":0.0,
        "MIO_NEAR_TOL_REL_GAP":1e-3,
        "MIO_REL_GAP_CONST":1e-10,
        "MIO_TOL_ABS_GAP":0.0,
        "MIO_TOL_ABS_RELAX_INT":1e-5,
        "MIO_TOL_FEAS":1e-6,
        "MIO_TOL_REL_DUAL_BOUND_IMPROVEMENT":0.0,
        "MIO_TOL_REL_GAP":1e-4,
        "MIO_TOL_X":1e-6,
        "OPTIMIZER_MAX_TIME":-1e+0,
        "PRESOLVE_TOL_ABS_LINDEP":1e-6,
        "PRESOLVE_TOL_AIJ":1e-12,
        "PRESOLVE_TOL_REL_LINDEP":1e-10,
        "PRESOLVE_TOL_S":1e-8,
        "PRESOLVE_TOL_X":1e-8,
        "QCQO_REFORMULATE_REL_DROP_TOL":1e-15,
        "SEMIDEFINITE_TOL_APPROX":1e-10,
        "SIM_LU_TOL_REL_PIV":1e-2,
        "SIMPLEX_ABS_TOL_PIV":1e-7,
        "UPPER_OBJ_CUT":1e+30,
        "UPPER_OBJ_CUT_FINITE_TRH":5e+29
    },
    "sparam":{
```


(continues on next page)

(continued from previous page)

```
            "BAS_SOL_FILE_NAME":"",
            "DATA_FILE_NAME":"examples/tools/data/lo1.mps",
            "DEBUG_FILE_NAME":"",
            "INT_SOL_FILE_NAME":"",
            "ITR_SOL_FILE_NAME":"",
            "MIO_DEBUG_STRING":"",
            "PARAM_COMMENT_SIGN":"%%",
            "PARAM_READ_FILE_NAME":"",
            "PARAM_WRITE_FILE_NAME":"",
            "READ_MPS_BOU_NAME":"",
            "READ_MPS_OBJ_NAME":"",
            "READ_MPS_RAN_NAME":"",
            "READ_MPS_RHS_NAME":"",
            "SENSITIVITY_FILE_NAME":"",
            "SENSITIVITY_RES_FILE_NAME":"",
            "SOL_FILTER_XC_LOW":"",
            "SOL_FILTER_XC_UPR":"",
            "SOL_FILTER_XX_LOW":"",
            "SOL_FILTER_XX_UPR":"",
            "STAT_FILE_NAME":"",
            "STAT_KEY":"",
            "STAT_NAME":"",
            "WRITE_LP_GEN_VAR_NAME":"XMSKGEN"
        }
    }
}
```

16.8 The Solution File Format

MOSEK provides several solution files depending on the problem type and the optimizer used:

- *basis solution file* (extension `.bas`) if the problem is optimized using the simplex optimizer or basis identification is performed,
- *interior solution file* (extension `.sol`) if a problem is optimized using the interior-point optimizer and no basis identification is required,
- *integer solution file* (extension `.int`) if the problem contains integer constrained variables.

All solution files have the format:

```
NAME                : <problem name>
PROBLEM STATUS      : <status of the problem>
SOLUTION STATUS     : <status of the solution>
OBJECTIVE NAME      : <name of the objective function>
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 status are shown. Next the primal and dual objective values are displayed.

- CONSTRAINTS For each constraint i of the form

$$l_i^c \leq \sum_{j=1}^{n} a_{ij} x_j \leq u_i^c, \tag{16.15}$$

the following information is listed:

  - INDEX: A sequential index assigned to the constraint by **MOSEK**
  - NAME: The name of the constraint assigned by the user.
  - AT: The status of the constraint. In Table 16.4 the possible values of the status keys and their interpretation are shown.

Table 16.4: Status keys.

Status key	Interpretation
UN	Unknown status
BS	Is basic
SB	Is superbasic
LL	Is at the lower limit (bound)
UL	Is at the upper limit (bound)
EQ	Lower limit is identical to upper limit
**	Is infeasible i.e. the lower limit is greater than the upper limit.

  - ACTIVITY: the quantity $\sum_{j=1}^{n} a_{ij} x_j^*$, where x^* is the value of the primal solution.
  - LOWER LIMIT: the quantity l_i^c (see (16.15).)
  - UPPER LIMIT: the quantity u_i^c (see (16.15).)
  - DUAL LOWER: the dual multiplier corresponding to the lower limit on the constraint.
  - DUAL UPPER: the dual multiplier corresponding to the upper limit on the constraint.

- VARIABLES The last section of the solution report lists information about the variables. This information has a similar interpretation as for the constraints. However, the column with the header CONIC DUAL is included for problems having one or more conic constraints. This column shows the dual variables corresponding to the conic constraints.

Example: lo1.sol

In Listing 16.7 we show the solution file for the lo1.opf problem.

Listing 16.7: An example of .sol file.

```
NAME                : 
PROBLEM STATUS      : PRIMAL_AND_DUAL_FEASIBLE
SOLUTION STATUS     : OPTIMAL
OBJECTIVE NAME      : obj
PRIMAL OBJECTIVE    : 8.33333333e+01
DUAL OBJECTIVE      : 8.33333332e+01

CONSTRAINTS
INDEX      NAME           AT ACTIVITY                 LOWER LIMIT        UPPER LIMIT        ␣
↪DUAL LOWER                DUAL UPPER
0          c1             EQ 3.00000000000000e+01     3.00000000e+01     3.00000000e+01     -0.
↪00000000000000e+00       -2.49999999741653e+00
1          c2             SB 5.33333333049187e+01     1.50000000e+01     NONE               2.
↪09159033069640e-10        -0.00000000000000e+00
2          c3             UL 2.49999999842049e+01     NONE               2.50000000e+01     -0.
↪00000000000000e+00       -3.33333332895108e-01

VARIABLES
INDEX      NAME           AT ACTIVITY                 LOWER LIMIT        UPPER LIMIT        ␣
↪DUAL LOWER                DUAL UPPER
0          x1             LL 1.67020427038537e-09     0.00000000e+00     NONE               -4.
↪49999999528054e+00       -0.00000000000000e+00
```

(continues on next page)

(continued from previous page)

```
1           x2             LL 2.93510446211883e-09      0.00000000e+00      1.00000000e+01      -2.
↪16666666494915e+00    6.20868657679896e-10
2           x3             SB 1.49999999899424e+01      0.00000000e+00      NONE                -8.
↪79123177245553e-10    -0.00000000000000e+00
3           x4             SB 8.33333332273115e+00      0.00000000e+00      NONE                -1.
↪69795978848200e-09    -0.00000000000000e+00
```

Chapter 17

List of examples

List of examples shipped in the distribution of Optimizer API for Python:

Table 17.1: List of distributed examples

File	Description
`blas_lapack.py`	Demonstrates the **MOSEK** interface to BLAS/LAPACK linear algebra routines
`callback.py`	An example of data/progress callback
`ceo1.py`	A simple conic exponential problem
`concurrent.py`	Implementation of a concurrent optimizer for linear and mixed-integer problems
`cqo1.py`	A simple conic quadratic problem
`feasrepairex1.py`	A simple example of how to repair an infeasible problem
`gp1.py`	A simple geometric program (GP) in conic form
`lo1.py`	A simple linear problem
`lo2.py`	A simple linear problem
`logistic.py`	Implements logistic regression and simple log-sum-exp (CEO)
`mico1.py`	A simple mixed-integer conic problem
`milo1.py`	A simple mixed-integer linear problem
`mioinitsol.py`	A simple mixed-integer linear problem with an initial guess
`modelLib.py`	Library of implementations of basic functions
`opt_server_async.py`	Uses **MOSEK** OptServer to solve an optimization problem asynchronously
`opt_server_sync.py`	Uses **MOSEK** OptServer to solve an optimization problem synchronously
`parallel.py`	Demonstrates parallel optimization
`parameters.py`	Shows how to set optimizer parameters and read information items
`portfolio_1_basic.py`	Portfolio optimization - basic Markowitz model
`portfolio_2_frontier.py`	Portfolio optimization - efficient frontier
`portfolio_3_impact.py`	Portfolio optimization - market impact costs
`portfolio_4_transcost.py`	Portfolio optimization - transaction costs
`portfolio_5_card.py`	Portfolio optimization - cardinality constraints
`pow1.py`	A simple power cone problem
`qcqo1.py`	A simple quadratically constrained quadratic problem
`qo1.py`	A simple quadratic problem
`reoptimization.py`	Demonstrate how to modify and re-optimize a linear problem
`response.py`	Demonstrates proper response handling

Continued on next page

Table 17.1 – continued from previous page

File	Description
`sdo1.py`	A simple semidefinite optimization problem
`sensitivity.py`	Sensitivity analysis performed on a small linear problem
`simple.py`	A simple I/O example: read problem from a file, solve and write solutions
`solutionquality.py`	Demonstrates how to examine the quality of a solution
`solvebasis.py`	Demonstrates solving a linear system with the basis matrix
`solvelinear.py`	Demonstrates solving a general linear system
`sparsecholesky.py`	Shows how to find a Cholesky factorization of a sparse matrix

Additional examples can be found on the **MOSEK** website and in other **MOSEK** publications.

Chapter 18

Interface changes

The section shows interface-specific changes to the **MOSEK** Optimizer API for Python in version 9.1 compared to version 8. See the release notes for general changes and new features of the **MOSEK** Optimization Suite.

18.1 Backwards compatibility

- **Parameters.** Users who set parameters to tune the performance and numerical properties of the solver (termination criteria, tolerances, solving primal or dual, presolve etc.) are recommended to reevaluate such tuning. It may be that other, or default, parameter settings will be more beneficial in the current version. The hints in Sec. 8 may be useful for some cases.
- All functions using the enum `accmode` were removed. Use corresponding separate functions for manipulating variables and constraints. For example, instead of

```
task.putbound(accmode.var, ...);
task.putbound(accmode.con, ...);
```

  use

```
task.putvarbound(...);
task.putconbound(...);
```

  and so on.
- Removed all `Near` problem and solution statuses i.e. `solsta.near_optimal`, `solsta.near_prim_infeas_cer`, etc. See Sec. 13.3.3.
- All functions related to the general nonlinear optimizer and Scopt have been removed. See Sec. 15.11.

18.2 Functions

Added

- *Env.setupthreads*
- *Task.appendsparsesymmatlist*
- *Task.generateconenames*
- *Task.generateconnames*
- *Task.generatevarnames*
- *Task.getacolslice*

- *Task.getacolslicenumnz*
- *Task.getarowslice*
- *Task.getarowslicenumnz*
- *Task.getatruncatetol*
- *Task.getbarsslice*
- *Task.getbarxslice*
- *Task.getclist*
- *Task.getskn*
- *Task.putatruncatetol*
- *Task.putbaraijlist*
- *Task.putbararowlist*
- *Task.putconboundlistconst*
- *Task.putconboundsliceconst*
- *Task.putconsolutioni*
- *Task.putvarboundlistconst*
- *Task.putvarboundsliceconst*
- *Task.putvarsolutionj*
- *Task.readjsonstring*
- *Task.readlpstring*
- *Task.readopfstring*
- *Task.readptfstring*

Removed

- `Task.checkconvexity`
- `Task.chgbound`
- `Task.getaslice`
- `Task.getaslicenumnz`
- `Task.getbound`
- `Task.getboundslice`
- `Task.getsolutioni`
- `Task.printdata`
- `Task.putbound`
- `Task.putboundlist`
- `Task.putboundslice`
- `Task.putsolutioni`

18.3 Parameters

Added

- *iparam.intpnt_order_gp_num_seeds*
- *iparam.intpnt_purify*
- *iparam.log_include_summary*
- *iparam.log_local_info*
- *iparam.mio_conic_outer_approximation*
- *iparam.mio_feaspump_level*
- *iparam.mio_max_num_root_cut_rounds*
- *iparam.mio_propagate_objective_constraint*
- *iparam.mio_seed*
- *iparam.opf_write_line_length*
- *iparam.presolve_max_num_pass*
- *iparam.ptf_write_transform*
- *iparam.sim_seed*
- *iparam.write_compression*

Removed

- dparam.data_tol_aij
- dparam.intpnt_nl_merit_bal
- dparam.intpnt_nl_tol_dfeas
- dparam.intpnt_nl_tol_mu_red
- dparam.intpnt_nl_tol_near_rel
- dparam.intpnt_nl_tol_pfeas
- dparam.intpnt_nl_tol_rel_gap
- dparam.intpnt_nl_tol_rel_step
- dparam.mio_disable_term_time
- dparam.mio_near_tol_abs_gap
- dparam.mio_near_tol_rel_gap
- iparam.mio_construct_sol
- iparam.mio_mt_user_cb
- iparam.opf_max_terms_per_line
- iparam.read_data_compressed
- iparam.read_data_format
- iparam.write_data_compressed
- iparam.write_data_format

18.4 Constants

Added

- *compresstype.zstd*
- *conetype.dexp*
- *conetype.dpow*
- *conetype.pexp*
- *conetype.ppow*
- *conetype.zero*
- *dataformat.ptf*
- *iinfitem.mio_numbin*
- *iinfitem.mio_numbinconevar*
- *iinfitem.mio_numcone*
- *iinfitem.mio_numconevar*
- *iinfitem.mio_numcont*
- *iinfitem.mio_numcontconevar*
- *iinfitem.mio_numdexpcones*
- *iinfitem.mio_numdpowcones*
- *iinfitem.mio_numintconevar*
- *iinfitem.mio_numpexpcones*
- *iinfitem.mio_numppowcones*
- *iinfitem.mio_numqcones*
- *iinfitem.mio_numrqcones*
- *iinfitem.mio_presolved_numbinconevar*
- *iinfitem.mio_presolved_numcone*
- *iinfitem.mio_presolved_numconevar*
- *iinfitem.mio_presolved_numcontconevar*
- *iinfitem.mio_presolved_numdexpcones*
- *iinfitem.mio_presolved_numdpowcones*
- *iinfitem.mio_presolved_numintconevar*
- *iinfitem.mio_presolved_numpexpcones*
- *iinfitem.mio_presolved_numppowcones*
- *iinfitem.mio_presolved_numqcones*
- *iinfitem.mio_presolved_numrqcones*
- *iinfitem.purify_dual_success*
- *iinfitem.purify_primal_success*
- *liinfitem.mio_anz*

Removed

- constant.dataformat.xml
- constant.dinfitem.mio_heuristic_time
- constant.dinfitem.mio_optimizer_time
- constant.iinfitem.mio_construct_num_roundings
- constant.iinfitem.mio_initial_solution
- constant.iinfitem.mio_near_absgap_satisfied
- constant.iinfitem.mio_near_relgap_satisfied
- constant.liinfitem.mio_sim_maxiter_setbacks
- constant.mionodeseltype.hybrid
- constant.mionodeseltype.worst
- constant.problemtype.geco
- constant.prosta.near_dual_feas
- constant.prosta.near_prim_and_dual_feas
- constant.prosta.near_prim_feas
- constant.sensitivitytype.optimal_partition
- constant.solsta.near_dual_feas
- constant.solsta.near_dual_infeas_cer
- constant.solsta.near_integer_optimal
- constant.solsta.near_optimal
- constant.solsta.near_prim_and_dual_feas
- constant.solsta.near_prim_feas
- constant.solsta.near_prim_infeas_cer

18.5 Response Codes

Added

- *rescode.err_appending_too_big_cone*
- *rescode.err_cbf_duplicate_pow_cones*
- *rescode.err_cbf_duplicate_pow_star_cones*
- *rescode.err_cbf_invalid_dimension_of_cones*
- *rescode.err_cbf_invalid_exp_dimension*
- *rescode.err_cbf_invalid_number_of_cones*
- *rescode.err_cbf_invalid_power*
- *rescode.err_cbf_invalid_power_cone_index*

- *rescode.err_cbf_invalid_power_star_cone_index*
- *rescode.err_cbf_power_cone_is_too_long*
- *rescode.err_cbf_power_cone_mismatch*
- *rescode.err_cbf_power_star_cone_mismatch*
- *rescode.err_cbf_unhandled_power_cone_type*
- *rescode.err_cbf_unhandled_power_star_cone_type*
- *rescode.err_cone_parameter*
- *rescode.err_format_string*
- *rescode.err_invalid_file_format_for_cfix*
- *rescode.err_invalid_file_format_for_free_constraints*
- *rescode.err_invalid_file_format_for_nonlinear*
- *rescode.err_invalid_file_format_for_ranged_constraints*
- *rescode.err_num_arguments*
- *rescode.err_ptf_format*
- *rescode.err_shape_is_too_large*
- *rescode.err_slice_size*
- *rescode.err_too_small_a_truncation_value*
- *rescode.wrn_exp_cones_with_variables_fixed_at_zero*
- *rescode.wrn_pow_cones_with_root_fixed_at_zero*

Removed

- rescode.err_cannot_clone_nl
- rescode.err_cannot_handle_nl
- rescode.err_invalid_accmode
- rescode.err_invalid_file_format_for_general_nl
- rescode.err_nonlinear_functions_not_allowed
- rescode.err_nr_arguments
- rescode.err_open_dl
- rescode.err_user_func_ret
- rescode.err_user_func_ret_data
- rescode.err_user_nlo_eval
- rescode.err_user_nlo_eval_hessubi
- rescode.err_user_nlo_eval_hessubj
- rescode.err_user_nlo_func
- rescode.trm_mio_near_abs_gap
- rescode.trm_mio_near_rel_gap
- rescode.wrn_construct_invalid_sol_itg
- rescode.wrn_construct_no_sol_itg
- rescode.wrn_construct_solution_infeas
- rescode.wrn_no_nonlinear_function_write

Bibliography

[AA95] E. D. Andersen and K. D. Andersen. Presolving in linear programming. *Math. Programming*, 71(2):221–245, 1995.

[AGMX96] E. D. Andersen, J. Gondzio, Cs. Mészáros, and X. Xu. Implementation of interior point methods for large scale linear programming. In T. Terlaky, editor, *Interior-point methods of mathematical programming*, pages 189–252. Kluwer Academic Publishers, 1996.

[ART03] E. D. Andersen, C. Roos, and T. Terlaky. On implementing a primal-dual interior-point method for conic quadratic optimization. *Math. Programming*, February 2003.

[AY96] E. D. Andersen and Y. Ye. Combining interior-point and pivoting algorithms. *Management Sci.*, 42(12):1719–1731, December 1996.

[And09] Erling D. Andersen. The homogeneous and self-dual model and algorithm for linear optimization. Technical Report TR-1-2009, MOSEK ApS, 2009. URL: http://docs.mosek.com/whitepapers/homolo.pdf.

[And13] Erling D. Andersen. On formulating quadratic functions in optimization models. Technical Report TR-1-2013, MOSEK ApS, 2013. Last revised 23-feb-2016. URL: http://docs.mosek.com/whitepapers/qmodel.pdf.

[BKVH07] S. Boyd, S.J. Kim, L. Vandenberghe, and A. Hassibi. A Tutorial on Geometric Programming. *Optimization and Engineering*, 8(1):67–127, 2007. Available at http://www.stanford.edu/~boyd/gp_tutorial.html.

[Chv83] V. Chvátal. *Linear programming.* W.H. Freeman and Company, 1983.

[GK00] Richard C. Grinold and Ronald N. Kahn. *Active portfolio management.* McGraw-Hill, New York, 2 edition, 2000.

[Naz87] J. L. Nazareth. *Computer Solution of Linear Programs.* Oxford University Press, New York, 1987.

[RTV97] C. Roos, T. Terlaky, and J. -Ph. Vial. *Theory and algorithms for linear optimization: an interior point approach.* John Wiley and Sons, New York, 1997.

[Ste98] G. W. Stewart. *Matrix Algorithms. Volume 1: Basic decompositions.* SIAM, 1998.

[Wal00] S. W. Wallace. Decision making under uncertainty: is sensitivity of any use. *Oper. Res.*, 48(1):20–25, January 2000.

[Wol98] L. A. Wolsey. *Integer programming.* John Wiley and Sons, 1998.

Symbol Index

Classes

Enumerations

Exceptions

Parameters

Response codes

Index