MOSEK Optimizer API for Python

Release 8.0.0.64

MOSEK ApS

2017

www.EngineeringBooksPdf.com

www.EngineeringBooksPdf.com

1 Introduction **1**
1.1 Why the Optimizer API for Python? . . . 1
1.2 License agreement . . . 1

2 Installation **3**
2.1 Compatibility . . . 3
2.2 Instructions . . . 3
2.3 Testing the Installation . . . 4

3 Basic Tutorials **5**
3.1 The Basics Tutorial . . . 6
3.2 Linear Optimization . . . 8
3.3 Conic Quadratic Optimization . . . 16
3.4 Semidefinite Optimization . . . 20
3.5 Quadratic Optimization . . . 24
3.6 Integer Optimization . . . 31
3.7 Optimizer Termination Handling . . . 35
3.8 Problem Modification and Reoptimization . . . 37
3.9 Solution Analysis . . . 41
3.10 Solver Parameters . . . 46

4 Nonlinear Tutorials **49**
4.1 Separable Convex (SCopt) Interface . . . 49

5 Advanced Tutorials **53**
5.1 The Progress Call-back . . . 53
5.2 Solving Linear Systems Involving the Basis Matrix . . . 55
5.3 Calling BLAS/LAPACK Routines from MOSEK . . . 62
5.4 Computing a Sparse Cholesky Factorization . . . 64
5.5 Converting a quadratically constrained problem to conic form . . . 69
5.6 **MOSEK** OptServer . . . 72

6 Guidelines **77**
6.1 Deployment . . . 77
6.2 Efficiency Considerations . . . 77
6.3 The license system . . . 79

7 Case Studies **81**
7.1 Portfolio Optimization . . . 81

8 Errors and Warnings **101**
8.1 Warnings . . . 101
8.2 Errors . . . 101

9 Managing I/O **103**
9.1 Stream I/O . . . 103

www.EngineeringBooksPdf.com

9.2 File I/O . . . 104
9.3 Verbosity . . . 104

10 Problem Formulation and Solutions **107**
10.1 Linear Optimization . . . 107
10.2 Conic Quadratic Optimization . . . 110
10.3 Semidefinite Optimization . . . 112
10.4 Quadratic and Quadratically Constrained Optimization . . . 114
10.5 General Convex Optimization . . . 115

11 The Optimizers for Continuous Problems **117**
11.1 Presolve . . . 117
11.2 Linear Optimization . . . 119
11.3 Conic Optimization . . . 125
11.4 Nonlinear Convex Optimization . . . 126
11.5 Using Multiple Threads in an Optimizer . . . 127

12 The Optimizer for Mixed-integer Problems **129**
12.1 Some Concepts and Facts Related to Mixed-integer Optimization . . . 129
12.2 The Mixed-integer Optimizer . . . 130
12.3 Termination Criterion . . . 130
12.4 Parameters Affecting the Termination of the Integer Optimizer. . . . 131
12.5 How to Speed Up the Solution Process . . . 131
12.6 Understanding Solution Quality . . . 132

13 Problem Analyzer **133**
13.1 General Characteristics . . . 134
13.2 Objective . . . 135
13.3 Linear Constraints . . . 135
13.4 Constraint and Variable Bounds . . . 136
13.5 Quadratic Constraints . . . 136
13.6 Conic Constraints . . . 136

14 Analyzing Infeasible Problems **137**
14.1 Example: Primal Infeasibility . . . 137
14.2 Locating the cause of Primal Infeasibility . . . 138
14.3 Locating the Cause of Dual Infeasibility . . . 139
14.4 The Infeasibility Report . . . 139
14.5 Theory Concerning Infeasible Problems . . . 141
14.6 The Certificate of Primal Infeasibility . . . 141
14.7 The certificate of dual infeasibility . . . 142

15 Sensitivity Analysis **143**
15.1 Sensitivity Analysis for Linear Problems . . . 143
15.2 Sensitivity Analysis with **MOSEK** . . . 149

16 API Reference **153**
16.1 API Conventions . . . 153
16.2 Functions grouped by topic . . . 157
16.3 The Interface for Separable Convex Optimization . . . 165
16.4 Parameters . . . 165
16.5 Response codes . . . 213
16.6 Enumerations . . . 235
16.7 Data Types . . . 261
16.8 Class Env . . . 262
16.9 Class Task . . . 268
16.10 Exceptions . . . 327

17 Supported File Formats **329**

www.EngineeringBooksPdf.com

17.1 The LP File Format . . . 330
17.2 The MPS File Format . . . 335
17.3 The OPF Format . . . 347
17.4 The CBF Format . . . 356
17.5 The XML (OSiL) Format . . . 371
17.6 The Task Format . . . 371
17.7 The JSON Format . . . 371
17.8 The Solution File Format . . . 379

18 Interface changes **381**
18.1 Compatibility . . . 381
18.2 Functions . . . 381
18.3 Parameters . . . 382
18.4 Constants . . . 384
18.5 Response Codes . . . 387

Bibliography **391**

API Index **393**

www.EngineeringBooksPdf.com

www.EngineeringBooksPdf.com

CHAPTER

ONE

INTRODUCTION

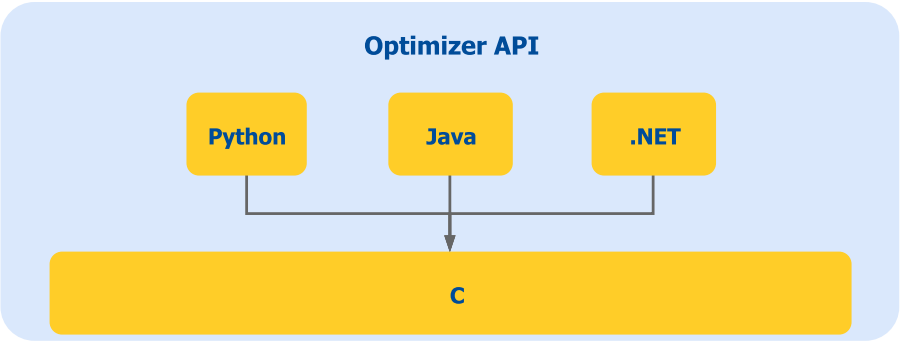
The **MOSEK** Optimization Suite 8.0.0.64 is a powerfull software package capable of solving large-scale optimization problems of the following kind:

- linear,
- convex quadratic,
- conic quadratic (also known as second-order cone),
- semidefinite,
- and general convex.

Integer constrained variables are supported for all problem classes except for semidefinite and general convex problems. In order to obtain an overview of features in the **MOSEK** Optimization Suite consult the product introduction guide.

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.

1.2 License agreement

Before using the **MOSEK** software, please read the license agreement available in the distribution at `<MSKHOME>/mosek/8/mosek-eula.pdf` or on the **MOSEK** website https://mosek.com/sales/license-agreement.

www.EngineeringBooksPdf.com

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 1.1.

Listing 1.1: *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 1.2.

Listing 1.2: *fplib* license.

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


www.EngineeringBooksPdf.com

CHAPTER

TWO

INSTALLATION

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

2.1 Compatibility

The **MOSEK** Optimizer API for Python requires Python with numpy. Below the supported Python versions are shown

Platform	Python	PyPy2.7
Linux 64 bit	2.7, 3.4 and newer	Yes
Mac OS 64 bit	2.7, 3.4 and newer	Yes
Windows 32 and 64 bit	2.7, 3.4 and newer	Yes

2.2 Instructions

2.2.1 Conda Install

Please follow the instructions in the Installation Guide.

2.2.2 Manual Install

Instructions for installing **MOSEK** Optimization Suite can in located in the Installation Guide.

Subsequently, let `<MSKHOME>` denote where **MOSEK** is installed

The relevant files of the **MOSEK** Optimizer API for Python are organized as reported in Table 2.1.

Table 2.1: Relevant files for the **MOSEK** Optimizer API for Python.

Relative Path	Description	Label
`<MSKHOME>/mosek/8/tools/platform/<PLATFORM>/python/2`	Python 2 install	*PYTHON2DIR*
`<MSKHOME>/mosek/8/tools/platform/<PLATFORM>/python/3`	Python 3 install	*PYTHON3DIR*
`<MSKHOME>/mosek/8/tools/examples/python`	Examples	*EXDIR*
`<MSKHOME>/mosek/8/tools/examples/data`	Additional data	*MISCDIR*

Install the Package

To install **MOSEK** run the `setup.py` script located in `<PYTHON2DIR>` or `<PYTHON3DIR>` depending on the Python version you want to use. For instance, to install **MOSEK** for Python 3 on a user level, i.e., with no special system privilegdes needed, type

www.EngineeringBooksPdf.com

```
$ 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.

2.2.3 PIP Install

A simple PIP installer is available on the **MOSEK** Github repository. Atlthough there is a **MOSEK** package in the official PIP repository then it is not created by **MOSEK**. It cannot be recommended to use that package.

2.3 Testing the Installation

The **MOSEK** Optimizer API for Python installation can be tested by running some of the enclosed examples. Indeed to run one of the distributed examples, open a terminal and change folder to `<EXDIR>` i.e.

```
cd <EXDIR>
```

then use your Python interpreter to run the example. For instance to execute example `lo1` type

```
python lo1.py
```


www.EngineeringBooksPdf.com

CHAPTER

THREE

BASIC TUTORIALS

In this section a number of examples is provided to demonstrate the functionality required for solving linear, conic, semidefinite and quadratic problems as well as mixed integer problems.

- *Basic tutorial* : This is the simplest tutorial: it solves a linear optimization problem read from file. It will show how
  - setup the **MOSEK** environment and problem task,
  - run the solver and
  - check the optimization results.
- *Linear optimization tutorial* : It shows how to input a linear program. It will show how
  - define variables and their bounds,
  - define constraints and their bounds,
  - define a linear objective function,
  - input a linear program but rows or by column.
  - retrieve the solution.
- *Conic quadratic optimization tutorial* : The basic steps needed to formulate a conic quadratic program are introduced:
  - define quadratic cones,
  - assign the relevant variables to their cones.
- *Semidefinite optimization tutorial* : How to input semidefintite optimization problems is the topic of this tutorial, and in particular how to
  - input semidefinite matrices and in sparse format,
  - add semidefinite matrix variable and
  - formulate linear constraints and objective function based on matrix variables.
- *Mixed-Integer optimization tutorial* : This tutorial shows how integrality conditions can be specified.
- *Quadratic optimization tutorial* : It shows how to input quadratic terms in the objective function and constraints.
- *Response code tutorial* : How to deal with the termination and solver status code is the topic of this tutorial:
  - what are termination and termination code,
  - how to check for errors and
  - which are the best practice to deal with them.

  This is a very important tutorial, every user should go through it.
- *Reoptimization tutorial* : This tutorial gives information on how to

www.EngineeringBooksPdf.com

  - modify linear constraints,
  - add new variables/constraints and
  - reoptimize the given problem, i.e. run the **MOSEK** optimizer again.

- *Solution analysis* : This tutorial shows how the user can analyze the solution returned by the solver.
- *Parameter setting tutorial* : This tutorial shows how to set the solver parameters.

3.1 The Basics Tutorial

The simplest program using the **MOSEK** Python interface can be described shortly:

1. Create an environment.
2. Set up environment specific data and initialize the environment.
3. Create a task.
4. Load a problem into the task.
5. Optimize the problem.
6. Fetch the result.
7. Delete the environment and task.

3.1.1 The environment and the task

The first **MOSEK** related step in any program that employs **MOSEK** is to create an environment object. The environment contains environment specific data such as information about the license file, streams for environment messages etc. When this is done one or more task objects can be created. Each task is associated with a single environment and defines a complete optimization problem as well as task message streams and optimization parameters.

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

```
# Create an environment
env = Env()

# Create a task
task = env.Task()

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

When done, tasks and environments may be disposed explicitly by calling the *dispose* method. This is not strictly necessary, but it will free up allocated resources and checked-out licenses immediately instead of when the garbage collector runs. From Python 2.6 and later the `with` construction can be used to dispose objects automatically when they drop of out of the `with` scope:

```
# Create an environment
with Env() as env:

    # Create a task
    with env.Task() as task:

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

Please note that multiple tasks should, if possible, share the same environment.

www.EngineeringBooksPdf.com

3.1.2 Example: Simple Working Example

The simple example in Listing 3.1 shows a working Python program which

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

Listing 3.1: A simple code solving a problem loaded from file.

```
import mosek
import sys

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

if len(sys.argv) <= 1:
    print ("Missing argument, syntax is:")
    print ("  simple inputfile [ solutionfile ]")
else:
    # 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')

        task.readdata (sys.argv[1])

        # Solve the problem
        task.optimize ()

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

        # If an output file was specified, write a solution
        if len(sys.argv) >= 3:
            # We define the output format to be OPF, and tell MOSEK to
            # leave out parameters and problem data from the output file.
            task.putintparam (mosek.iparam.write_data_format,    mosek.dataformat.op)
            task.putintparam (mosek.iparam.opf_write_solutions,  mosek.onoffkey.on)
            task.putintparam (mosek.iparam.opf_write_hints,      mosek.onoffkey.off)
            task.putintparam (mosek.iparam.opf_write_parameters, mosek.onoffkey.off)
            task.putintparam (mosek.iparam.opf_write_problem,    mosek.onoffkey.off)

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

Reading and Writing Problems

Use the *task.writedata* function to write a problem to a file. By default, when not choosing any specific file format for the parameter *iparam.write_data_format*, **MOSEK** will determine the output file format by the extension of the file name:

www.EngineeringBooksPdf.com

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

Similarly, controlled by *iparam.read_data_format*, the function *task.readdata* can read a problem from a file:

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

Working with the problem data

An optimization problem consists of several components; objective, objective sense, constraints, variable bounds etc.

Therefore, the interface provides a number of methods to operate on the task specific data, all of which are listed under the *Task* class-specification.

Setting parameters

Apart from the problem data, the task contains a number of parameters defining the behavior of **MOSEK**. For example the *iparam.optimizer* parameter defines which optimizer to use. There are three kinds of parameters in **MOSEK**

- Integer parameters that can be set with *task.putintparam*,
- Double parameters that can be set with *task.putdouparam*, and
- string parameters that can be set with *task.putstrparam*,

The values for integer parameters are either simple integer values or enum values. See Section *3.10* for more details on how to set parameters.

A complete list of all parameters is found in Section *16.4*.

3.2 Linear Optimization

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

Minimize or maximize the objective function

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

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

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

where we have used the problem elements:

- m and n which are the number of constraints and variables respectively,
- x which is the variable vector of length n,
- c which is a coefficient vector of size n

www.EngineeringBooksPdf.com

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

- c^f which is a constant,
- A which is a $m \times n$ matrix of coefficients is given by

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

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

Note: Please note the unconventional notation using 0 as the first index rather than 1. Hence, x_0 is the first element in variable vector x.

3.2.1 Example LO1

The following is an example of a linear optimization problem:

$$\begin{array}{lccccccccc} \text{maximize} & 3x_0 & + & 1x_1 & + & 5x_2 & + & 1x_3 & & \\ \text{subject to} & 3x_0 & + & 1x_1 & + & 2x_2 & & & = & 30, \\ & 2x_0 & + & 1x_1 & + & 3x_2 & + & 1x_3 & \geq & 15, \\ & & & 2x_1 & & & + & 3x_3 & \leq & 25, \end{array} \tag{3.1}$$

having the bounds

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

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.

www.EngineeringBooksPdf.com

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.

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])
```

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*.

www.EngineeringBooksPdf.com

```
# 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` specify the type of the bound according to Table 3.1.

Table 3.1: 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

Interpretation of the bound keys.

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

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

and

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

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 index 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.
```

Alternatively, the same A matrix can be set one row at a time; please see Listing 3.3.

Finally, the bounds on each constraint are set by looping over each constraint index $i = 0, \ldots,$ `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])
 task.putconboundslice(0,numcon, bkc,blc,buc);
```


www.EngineeringBooksPdf.com

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* or *solsta.near_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 cache any exceptions thrown by **MOSEK** in the lines:

```
except mosek.Exception 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 *16.5*.

Source code for lo1

Listing 3.2: Linear optimization example: complete listing.

```
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
```


www.EngineeringBooksPdf.com

```
    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)
    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])
    task.putconboundslice(0,numcon, bkc,blc,buc);
```


www.EngineeringBooksPdf.com

```
      # 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 or
          solsta == mosek.solsta.near_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 or
            solsta == mosek.solsta.near_dual_infeas_cer or
            solsta == mosek.solsta.near_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.Exception 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)
sys.exit(0)
```

Row-wise input

In the previous example the A matrix is set one column at a time. Alternatively the same matrix can be set one row at a time or the two methods can be mixed as in the example in Section *3.8*. The following example show how to set the A matrix by rows

Listing 3.3: Example showing how to input the A matrix row-wise.

```
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):
```


www.EngineeringBooksPdf.com

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

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

      # 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 ]

      # We input the A matrix column-wise
      # asub contains row indexes
      asub = [ [0, 1, 2],
               [0, 1, 2, 3],
               [0, 3]]
      # acof contains coefficients
      aval = [ [3.0, 1.0, 2.0],
               [2.0, 1.0, 3.0, 1.0],
               [2.0, 3.0]]
      numvar = len(bkx)
      numcon = len(bkc)
      # Append 'numcon' empty constraints.
      # The constraints will initially have no bounds.
      task.appendcons(numcon)

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

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

      for i in range(numcon):
        task.putbound(mosek.accmode.con,i,bkc[i],blc[i],buc[i])
```


www.EngineeringBooksPdf.com

```
        # Input row i of A
        task.putarow(i,                         # Row index.
                     asub[i],                   # Column indexes of non-zeros in row i.
                     aval[i]);                  # Non-zero Values of row i.

      # Input the objective sense (minimize/maximize)
      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.bas)
      solsta = task.getsolsta(mosek.soltype.bas)

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

      if solsta == mosek.solsta.optimal or solsta == mosek.solsta.near_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 solsta == mosek.solsta.near_dual_infeas_cer:
          print("Primal or dual infeasibility.\n")
      elif  solsta == mosek.solsta.near_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.Exception 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)
sys.exit(0)
```

3.3 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. Actually, since the set $\mathbb{R}^n$ of real numbers is also a convex cone, all variables can in fact be partitioned into subsets belonging to separate convex cones, simply stated $x \in \mathcal{K}$.

www.EngineeringBooksPdf.com

MOSEK can solve conic quadratic 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

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

For convenience, the user only specify subsets of variables x^t belonging to cones $\mathcal{K}_t$ different from the set $\mathbb{R}^{n_t}$ of real numbers. These cones can be a:

- 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:

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

From these definition it follows that

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

is equivalent to

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

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.

3.3.1 Example CQO1

We want to solve the following Conic Optimization Problem problem:

$$\begin{array}{lccc} \text{minimize} & x_4 + x_5 + x_6 & & \\ \text{subject to} & x_1 + x_2 + 2x_3 & = & 1, \\ & x_1, x_2, x_3 & \geq & 0, \\ & x_4 \geq \sqrt{x_1^2 + x_2^2}, & & \\ & 2x_5 x_6 \geq x_3^2 & & \end{array} \tag{3.2}$$

is an example of a conic quadratic optimization problem. The problem involves some linear constraints, a quadratic cone and a rotated quadratic cone.

Implementation

Problem (3.2) can be implemeted using the **MOSEK** Python API as follows:

www.EngineeringBooksPdf.com

Listing 3.4: Source code solving problem (3.2).

```
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()

# We might write everything directly as a script, but it looks nicer
# to create a function.
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.putbound(mosek.accmode.var,j,bkx[j],blx[j],bux[j])
```


www.EngineeringBooksPdf.com

```
      for j in range(len(aval)):
        # 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.
      for i in range(numcon):
          task.putbound(mosek.accmode.con,i,bkc[i],blc[i],buc[i])

          # Input the cones
      task.appendcone(mosek.conetype.quad,
                      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 or solsta == mosek.solsta.near_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 solsta == mosek.solsta.near_dual_infeas_cer:
        print("Primal or dual infeasibility.\n")
      elif solsta == mosek.solsta.near_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.Exception 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)
sys.exit(0)
```


www.EngineeringBooksPdf.com

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

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

The first argument selects the type of quadratic cone. Either *conetype.quad* for a *quadratic cone* or *conetype.rquad* for a *rotated quadratic cone*. The cone parameter *0.0* is currently not used by **MOSEK** — simply passing *0.0* will work.

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

3.4 Semidefinite Optimization

Semidefinite optimization is a generalization of conic quadratic 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

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

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

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

3.4.1 Example SDO1

The problem

$$\begin{array}{lrcl} \text{minimize} & \left\langle \begin{bmatrix} 2 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 2 \end{bmatrix}, \overline{X} \right\rangle + x_0 & & \\ \text{subject to} & \left\langle \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \overline{X} \right\rangle + x_0 & = & 1, \\ & \left\langle \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \overline{X} \right\rangle + x_1 + x_2 & = & 1/2, \\ & x_0 \geq \sqrt{{x_1}^2 + {x_2}^2}, & \overline{X} \succeq 0, & \end{array} \tag{3.3}$$

is a mixed semidefinite and conic quadratic programming problem with a 3-dimensional semidefinite variable

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

www.EngineeringBooksPdf.com

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

$$\overline{X}_{00} + \overline{X}_{11} + \overline{X}_{22} + x_0 = 1,$$

and

$$\overline{X}_{00} + \overline{X}_{11} + \overline{X}_{22} + 2(\overline{X}_{10} + \overline{X}_{20} + \overline{X}_{21}) + x_1 + x_2 = 1/2.$$

Problem (3.3) is implemented in Listing 3.5.

Listing 3.5: Source code solving problem (3.3).

```
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]
      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],
```


www.EngineeringBooksPdf.com

```
            [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 j.
             aval[i])             # Non-zero values of row j.

task.appendcone(mosek.conetype.quad,
                0.0,
                conesub)

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)
```


www.EngineeringBooksPdf.com

```
      # 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 or
          solsta == mosek.solsta.near_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 or
            solsta == mosek.solsta.near_dual_infeas_cer or
            solsta == mosek.solsta.near_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.Exception 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)
sys.exit(0)
```

Source code comments

This example introduces several new functions. The first new function *task.appendbarvars* is used to append the semidefinite variable:

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

Symmetric matrices are created using the function *task.appendsparsesymmat*:

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

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


www.EngineeringBooksPdf.com

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

The second argument specifies the dimension of the symmetric variable and the third argument gives the number of non-zeros in the lower triangular part of the matrix. The next three arguments specify the non-zeros in the lower-triangle in triplet format, and the last argument will be updated with a unique index of the created symmetric matrix.

After one or more symmetric matrices have been created using *task.appendsparsesymmat*, we can combine them to setup a objective matrix coefficient $\overline{C}_j$ using *task.putbarcj*, which forms a linear combination of one more symmetric matrices:

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

The second argument specify the semidefinite variable index j; in this example there is only a single variable, so the index is 0. The next three arguments give the number of matrices used in the linear combination, their indices (as returned by *task.appendsparsesymmat*), and the weights for the individual matrices, respectively. In this example, we form the objective matrix coefficient directly from a single symmetric matrix.

Similary, a constraint matrix coefficient $\overline{A}_{ij}$ is setup by the function *task.putbaraij*:

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

where the second argument specifies the constraint number (the corresponding row of $\overline{A}$), and the third argument specifies the semidefinite variable index (the corresponding column of $\overline{A}$). The next three arguments specify a weighted combination of symmetric matrices used to form the constraint matrix coefficient. 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 given in the second argument, and the third argument is a pointer to an array for storing the numerical values.

3.5 Quadratic Optimization

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

$$\begin{array}{lccccll} \text{minimize} & & & \frac{1}{2}x^T Q^o x + c^T x + c^f & & & \\ \text{subject to} & l_k^c & \leq & \frac{1}{2}x^T Q^k x + \sum_{j=0}^{n-1} a_{k,j} x_j & \leq & u_k^c, & k = 0, \ldots, m-1, \\ & l_j^x & \leq & x_j & \leq & u_j^x, & j = 0, \ldots, n-1. \end{array} \tag{3.4}$$

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

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

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

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

$$l_k^c \leq \frac{1}{2} x^T Q^k x + \sum_{j=0}^{n-1} a_{k,j} x_j \tag{3.5}$$

www.EngineeringBooksPdf.com

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

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

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

A matrix is positive semidefinite if the smallest eigenvalue of the matrix is nonnegative. An alternative statement of the positive semidefinite requirement is

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

If Q is not positive semidefinite, then **MOSEK** will not produce reliable results or work at all.

One way of checking whether Q is positive semidefinite is to check whether all the eigenvalues of Q are nonnegative.

3.5.1 Example: Quadratic Objective

$$\begin{array}{lll} \text{minimize} & & x_1^2 + 0.1x_2^2 + x_3^2 - x_1x_3 - x_2 \\ \text{subject to} & 1 \leq & x_1 + x_2 + x_3 \\ & & x \geq 0. \end{array} \tag{3.6}$$

For the example (3.6) implies that

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

and that

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

Please note the explicit $\frac{1}{2}$ in the objective function of (3.4) which implies that diagonal elements must be doubled in Q, i.e. $Q_{11} = 2$, whereas the coefficient in (3.6) is 1 in front of x_1^2.

Important: **MOSEK** assumes that the Q matrix is symmetric, i.e. $Q = Q^T$, and that Q is *positive semidefinite*.

The source code follows in Listing 3.6.

Listing 3.6: Source code implementing problem (3.6).

```
import sys
import os

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()
```


www.EngineeringBooksPdf.com

```
# We might write everything directly as a script, but it looks nicer
# to create a function.
def main ():
  # Open MOSEK and create an environment and task
  # 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() 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.putbound(mosek.accmode.var,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.
      for i in range(numcon):
         task.putbound(mosek.accmode.con,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)
```


www.EngineeringBooksPdf.com

```
      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 or solsta == mosek.solsta.near_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 solsta == mosek.solsta.near_dual_infeas_cer:
        print("Primal or dual infeasibility.\n")
      elif  solsta == mosek.solsta.near_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.Exception 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)
print ("Finished OK")
sys.exit(0)
```

Example code comments

Most of the functionality in this example has already been explained for the linear optimization example in Section *3.2* and it will not be repeated here.

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

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

$$Q^o_{ij} = Q^o_{ji}.$$

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

The lower triangular part of the matrix Q^o is specified using an unordered sparse triplet format (for details, see Section *16.1.3*):

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


www.EngineeringBooksPdf.com

Please note that

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

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

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

3.5.2 Example: Quadratic constraints

In this section describes how to solve a problem with quadratic constraints. Please note that quadratic constraints are subject to the convexity requirement (3.5).

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

$$\begin{array}{lrcl} \text{minimize} & \frac{1}{2}x^T Q^o x + c^T x & & \\ \text{subject to} & \frac{1}{2}x^T Q^0 x + Ax & \geq & b, \end{array} \tag{3.7}$$

where

$$Q^o = \begin{bmatrix} 2 & 0 & -1 \\ 0 & 0.2 & 0 \\ -1 & 0 & 2 \end{bmatrix}, c = \begin{bmatrix} 0 & -10 \end{bmatrix}, A = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}, b = 1.$$

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

Listing 3.7: Script implementing problem (3.7).

```
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()

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


www.EngineeringBooksPdf.com

```
    # Attach a printer to the task
    task.set_Stream (mosek.streamtype.log, streamprinter)


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

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

    c     = [ 0.0, -1.0, 0.0 ]

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

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

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

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

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

    for i in range(numcon):
        task.putbound(mosek.accmode.con,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   ]
```


www.EngineeringBooksPdf.com

```
      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,
                 xx)

      if solsta == mosek.solsta.optimal or solsta == mosek.solsta.near_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 solsta == mosek.solsta.near_dual_infeas_cer:
        print("Primal or dual infeasibility.\n")
      elif  solsta == mosek.solsta.near_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.Exception as e:
    print ("ERROR: %s" % str(code))
    if msg is not None:
        print ("\t%s" % e.msg)
        sys.exit(1)
except:
    import traceback
    traceback.print_exc()
    sys.exit(1)
sys.exit(0)
```

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

www.EngineeringBooksPdf.com

3.6 Integer Optimization

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

Section *3.6.2* shows *how to input an initial feasible solution* to help the solver.

3.6.1 Example MILO1

In this section the example

$$\begin{array}{lrcl} \text{maximize} & x_0 + 0.64x_1 & & \\ \text{subject to} & 50x_0 + 31x_1 & \leq & 250, \\ & 3x_0 - 2x_1 & \geq & -4, \\ & x_0, x_1 \geq 0 & & \text{and integer} \end{array} \tag{3.8}$$

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

The example (3.8) is almost identical to a linear optimization problem (see *3.2*) except for some variables being integer constrained. Therefore, only the specification of the integer constraints requires something new compared to the linear optimization problem discussed previously.

In **MOSEK** these constraints are specified using the function *task.putvartype* as shown in the code:

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

The complete source for the example is listed Listing 3.8. 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 3.8: Source code implementing problem (3.8).

```
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()

# We might write everything directly as a script, but it looks nicer
# to create a function.
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.up, mosek.boundkey.lo  ]
      blc = [              -inf,              -4.0  ]
      buc = [             250.0,               inf  ]
```


www.EngineeringBooksPdf.com

```
    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.
                   asub[j],             # Row index of non-zeros in column 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 ])

    # 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)

    # X

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

    if solsta in [ mosek.solsta.integer_optimal, mosek.solsta.near_integer_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:
```


www.EngineeringBooksPdf.com

```
        print("Primal or dual infeasibility.\n")
      elif solsta == mosek.solsta.near_dual_infeas_cer:
        print("Primal or dual infeasibility.\n")
      elif  solsta == mosek.solsta.near_prim_infeas_cer:
        print("Primal or dual infeasibility.\n")
      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.Exception 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)
sys.exit(0)
```

Solving a mixed-integer optimization program could easily result in long running time. It is therefore of interest to consider a termination criterion based on the maximum running time. This is possible setting the *dparam.mio_max_time*. See Section *3.10* for more details on how to set solver parameters.

3.6.2 Specifying an initial solution

Integer optimization problems are generally hard to solve, but the solution time can often be reduced by providing an initial solution for the solver. It is not necessary to specify the whole solution. By setting the *iparam.mio_construct_sol* parameter to *onoffkey.on* and inputting values for the integer variables only, will force **MOSEK** to compute the remaining continuous variable values.

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

Consider the problem

$$\begin{array}{ll} \text{maximize} & 7x_0 + 10x_1 + x_2 + 5x_3 \\ \text{subject to} & x_0 + x_1 + x_2 + x_3 \leq 2.5 \\ & x_0, x_1, x_2 \in \mathbb{Z} \\ x_0, x_1, x_2, x_3 \geq 0 & \end{array} \tag{3.9}$$

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

Solution values can be set using *task.putsolution* (for inputting a whole solution) or *task.putsolutioni* (for inputting solution values related to a single variable or constraint).

Listing 3.9: Implementation of problem (3.9) specifying an initial solution.

```
import sys
import mosek
```


www.EngineeringBooksPdf.com

```
# 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()


# We might write everything directly as a script, but it looks nicer
# to create a function.
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.up ]
        blc  = [ -inf,             ]
        buc  = [ 2.5               ]

        bkx  = [ mosek.boundkey.lo,
                 mosek.boundkey.lo,
                 mosek.boundkey.lo,
                 mosek.boundkey.lo ]

        blx  = [0.0, 0.0, 0.0, 0.0 ]
        bux  = [ inf, inf,  inf,  inf ]

        c    = [ 7.0, 10.0, 1.0, 5.0 ]

        asub = [  0,   0,   0,   0   ]
        acof = [  1.0, 1.0, 1.0, 1.0]

        ptrb = [ 0, 1, 2, 3 ]
        ptre = [ 1, 2, 3, 4 ]

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

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

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

        # Define variables to be integers

        task.putvartypelist([ 0, 1, 2 ],
```


www.EngineeringBooksPdf.com

```
                            [ mosek.variabletype.type_int,
                              mosek.variabletype.type_int,
                              mosek.variabletype.type_int])

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

        # Assign values 0,2,0 to integer variables. Important to
        # assign a value to all integer constrained variables.
        task.putxxslice(mosek.soltype.itg,0,3,[0.0, 2.0, 0.0])

        # Optimize
        task.optimize()

        # Did mosek construct a feasible initial solution ?
        if task.getintinf(mosek.iinfitem.mio_construct_solution) > 0:
            print("Objective value of constructed integer solution: %-24.12e" % task.
↪getdouinf(mosek.dinfitem.mio_construct_solution_obj))
        else:
            print("Intial integer solution construction failed.");

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

            # Output a solution
            xx = [0.]*numvar
            task.getxx(mosek.soltype.itg, xx)
            print("Integer optimal solution")
            for j in range(0,numvar) :
                print("\tx[%d] = %e" % (j,xx[j]))
        else:
            print("No integer solution is available.")

# call the main function
try:
    main ()
except mosek.Exception 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)
```

3.7 Optimizer Termination Handling

After solving an optimization problem with **MOSEK** an approriate action must be taken depending on the outcome. Usually the expected outcome is an optimal solution, but there may be several situations where this is not the result. E.g., if the problem is infeasible or nearly so or if the solver ran out of memory or stalled while optimizing, the result may not be as expected.

This section discusses what should be considered when an optimization has ended unsuccessfully.

Before continuing, let us consider the four status codes available in **MOSEK** that is relevant for the error handing:

- **Termination code**: It provides information about why the optimizer terminated. For instance if

www.EngineeringBooksPdf.com

a time limit has been specfied (this is common for mixed integer problems), the termination code will tell if this termination limit was the cause of the termination. Note that reaching a prespecfied time limit is not considered an exceptional case. It must be expected that this occurs occasionally.

- **Response code**: It is an information about the system status and the outcome of the call to a **MOSEK** functionalities. This code is used to report the unexpected failures such as out of space.

MOSEK runs silently when no errors are encountered, while an exception is generated otherwise. See *16.5* for a list of possible exceptions.

- **Solution status**: It contains information about the status of the solution, e.g., whether the solution is optimal or a certificate of infeasibility.
- **Problem status**: It describes what **MOSEK** knows about the feasibility of the problem, i.e., if the is problem feasible or infeasible.

The problem status is mostly used for integer problems. For continuous problems a problem status of, say, *infeasible* will always mean that the solution is a certificate of infeasibility. For integer problems it is not possible to provide a certificate, and thus a separate problem status is useful.

Note that if we want to report, e.g., that the optimizer terminated due to a time limit or because it stalled but with a feasible solution, we have to consider *both* the termination code, *and* the solution status.

The following pseudo code demonstrates a best practice way of dealing with the status codes.

- `if ( the solution status is as expected )`
  - **The normal case**:

    Do whatever that was planned. Note the response code is ignored because the solution has the expected status. Of course we may check the response anyway if we like.
- `else`
  - **Exceptional case**:

    Based on solution status, response and termination codes take appropriate action.

In Listing 3.10 the pseudo code is implemented. The idea of the example is to read an optimization problem from a file, e.g., an MPS file and optimize it. Based on status codes an appropriate action is taken, which in this case is to print a suitable message.

Listing 3.10: A typical code that handle **MOSEK** response code.

```
import mosek
import sys

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

def main(args):
  if len(args) < 1:
    print ("No input file specified")
    return
  else:
    print ("Inputfile: %s" % args[0])

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

        task.readdata(args[0])

        trmcode = task.optimize()
```


www.EngineeringBooksPdf.com

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

        if solsta in [ mosek.solsta.optimal,
                       mosek.solsta.near_optimal ]:
            print ("An optimal basic solution is located.")
            task.solutionsummary(mosek.streamtype.log)
        elif solsta in [ mosek.solsta.dual_infeas_cer,
                         mosek.solsta.near_dual_infeas_cer ]:
            print ("Dual infeasibility certificate found.")
        elif solsta in [ mosek.solsta.prim_infeas_cerl,
                         mosek.solsta.near_prim_infeas_cer ]:
            print("Primal infeasibility certificate found.\n");
        elif solsta == mosek.solsta.sta_unknown:
            # The solutions status is unknown. The termination code
            # indicating why the optimizer terminated prematurely.
            print ("The solution status is unknown.")
            print ("Termination code: %s" % str(trmcode))
        else:
            print ("An unexpected solution status is obtained.")

  except mosek.MosekException as err:
    print ("Generic error:")
    print (err)

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

3.8 Problem Modification and Reoptimization

Often one might want to solve not just a single optimization problem, but a sequence of problem, each differing only slightly from the previous one. This section demonstrates how to modify and 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 three categories:

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

These operations may be costly and, especially removing variables and constraints. 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.

For instance the former optimal solution may be still feasibile, but no more optimal; or for tiny modifications of the objective function it may be still optimal. This is a special case that we discuss in Section *15*.

In general, **MOSEK** exploits dual information and the availablity 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 reliable and effective as those for the simplex algorithm. More information can be found in Chapter 10 of the book *[Chv83]*.

www.EngineeringBooksPdf.com

3.8.1 Example: Production Planning

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

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

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

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

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

$$\begin{array}{lrcrcrcl} \text{maximize} & 1.5x_0 & + & 2.5x_1 & + & 3.0x_2 & & \\ \text{subject to} & 2x_0 & + & 4x_1 & + & 3x_2 & \leq & 100000, \\ & 3x_0 & + & 2x_1 & + & 3x_2 & \leq & 50000, \\ & 2x_0 & + & 3x_1 & + & 2x_2 & \leq & 60000, \end{array} \tag{3.10}$$

and

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

Code in Listing 3.11 loads and solves this problem:

Listing 3.11: How to load problem (3.10)

```
# Create a MOSEK environment
with mosek.Env () as env:

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

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

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

    # Objective coefficients
    csub = [   0,   1,   2 ]
    cval = [ 1.5, 2.5, 3.0 ]

    # We input the A matrix column-wise
    # asub contains row indexes
    asub = [ 0, 1, 2,
```


www.EngineeringBooksPdf.com

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

3.8.2 Changing the A Matrix

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

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


www.EngineeringBooksPdf.com

The problem now has the form:

$$\begin{array}{lccccccc} \text{maximize} & 1.5x_0 & + & 2.5x_1 & + & 3.0x_2 & & \\ \text{subject to} & 3x_0 & + & 4x_1 & + & 3x_2 & \leq & 100000, \\ & 3x_0 & + & 2x_1 & + & 3x_2 & \leq & 50000, \\ & 2x_0 & + & 3x_1 & + & 2x_2 & \leq & 60000, \end{array} \tag{3.11}$$

and

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

After changing the A matrix we can find the new optimal solution by calling *task.optimize* again

3.8.3 Appending Variables

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

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

This corresponds to creating a new variable x_3 , appending a new column to the A matrix and setting a new value in the objective. We do this in Listing 3.12

Listing 3.12: How to add a column.

```
# Append a new varaible x_3 to the problem */
task.appendvars(1)

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

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

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

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

After this operation the problem looks this way:

$$\begin{array}{lccccccccc} \text{maximize} & 1.5x_0 & + & 2.5x_1 & + & 3.0x_2 & + & 1.0x_3 & & \\ \text{subject to} & 3x_0 & + & 4x_1 & + & 3x_2 & + & 4x_3 & \leq & 100000, \\ & 3x_0 & + & 2x_1 & + & 3x_2 & & & \leq & 50000, \\ & 2x_0 & + & 3x_1 & + & 2x_2 & + & 1x_3 & \leq & 60000, \end{array} \tag{3.12}$$

and

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

3.8.4 Reoptimization

When *task.optimize* is called **MOSEK** will store the optimal solution internally. After a task has been modified and *task.optimize* is called again the solution will automatically be used to reduce solution time of the new problem, if possible.

www.EngineeringBooksPdf.com

In this case an optimal solution to problem (3.11) was found and then added a column was added to get (3.12). We let **MOSEK** select the suitable simplex algorithm to perform reoptimization.

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

3.8.5 Appending Constraints

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

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

This corresponds to adding the constraint

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

to the problem which is done in the following code:

```
# Append a new constraint
task.appendcons(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)
```

3.9 Solution Analysis

The main purpose of **MOSEK** is to solve optimization problems and therefore the most fundamental question to be asked is whether the solution reported by **MOSEK** is a solution to the desired optimization problem.

There can be several reasons why it might be not case. The most prominent reasons are:

- A wrong problem. The problem inputted to **MOSEK** is simply not the right problem, i.e. some of the data may have been corrupted or the model has been incorrectly built.
- Numerical issues. The problem is badly scaled or otherwise badly posed.
- Other reasons. E.g. not enough memory or an explicit user request to stop.

The first step in verifying that **MOSEK** reports the expected solution is to inspect the solution summary generated by **MOSEK** (see Section *3.9.1*). The solution summary provides information about

- the problem and solution statuses,
- objective value and infeasibility measures for the primal solution, and

www.EngineeringBooksPdf.com

- objective value and infeasibility measures for the dual solution, where applicable.

By inspecting the solution summary it can be verified that **MOSEK** produces a feasible solution, and, in the continuous case, the optimality can be checked using the dual solution. Furthermore, the problem itself ca be inspected using the problem analyzer discussed in Section *13*.

If the summary reports conflicting information (e.g. a solution status that does not match the actual solution), or the cause for terminating the solver before a solution was found cannot be traced back to the reasons stated above, it may be caused by a bug in the solver; in this case, please contact **MOSEK** support (see Section *1.2*).

If it has been verified that **MOSEK** solves the problem correctly but the solution is still not as expected, next step is to verify that the primal solution satisfies all the constraints. Hence, using the original problem it must be determined whether the solution satisfies all the required constraints in the model. For instance assume that the problem has the constraints

$$\begin{array}{rcl} x_1 + 2x_2 + x_3 & \leq & 1, \\ x_1, x_2, x_3 \geq 0 & & \end{array}$$

and **MOSEK** reports the optimal solution

$$x_1 = x_2 = x_3 = 1.$$

Then clearly the solution violates the constraints. The most likely explanation is that the model does not match the problem entered into **MOSEK**, for instance

$$x_1 - 2x_2 + x_3 \leq 1$$

may have been inputted instead of

$$x_1 + 2x_2 + x_3 \leq 1.$$

A good way to debug such an issue is to dump the problem to *OPF file* and check whether the violated constraint has been specified correctly.

Verifying that a feasible solution is optimal can be harder. However, for continuous problems, i.e. problems without any integer constraints, optimality can verified using a dual solution. Normally, **MOSEK** will report a dual solution; if that is feasible and has the same objective value as the primal solution, then the primal solution must be optimal.

An alternative method is to find another primal solution that has better objective value than the one reported to **MOSEK**. If that is possible then either the problem is badly posed or there is bug in **MOSEK**.

3.9.1 The Solution Summary

Due to **MOSEK** employs finite precision floating point numbers then reported solution is an approximate optimal solution. Therefore after solving an optimization problem it is relevant to investigate how good an approximation the solution is. For a convex optimization problem that is an easy task because the optimality conditions are:

- The primal solution must satisfy all the primal constraints.
- The dual solution much satisfy all the dual constraints.
- The primal and dual objective values must be identical.

Therefore, the **MOSEK** solution summary displays that information that makes it possible to verify the optimality conditions. Indeed the solution summary reports how much primal and dual solutions violate the primal and constraints respectively. In addition the objective values assoctaied with each solution repoted.

In case of a linear optimization problem the solution summary may look like

www.EngineeringBooksPdf.com

```
Basic solution summary
  Problem status  : PRIMAL_AND_DUAL_FEASIBLE
  Solution status : OPTIMAL
  Primal.  obj: -4.6475314286e+002  nrm: 5e+002   Viol.  con: 1e-014   var: 1e-014
  Dual.    obj: -4.6475314543e+002  nrm: 1e+001   Viol.  con: 4e-009   var: 4e-016
```

The interpreation of the solution summary is as follows:

- Information for the basic solution is reported.
- The problem status is primal and dual feasible which means the problem has an optimal solution.
- The solution status is optimal.
- Next information about the primal solution is reported. The information consists of the objective value, the infinity norm of the primal solution and violation meassures. The violation for the constraints (`con:`) is the maximal violation in any of the constraints. Whereas the violations for the variables (`var:`) is the maximal bound violation for any of the variables. In this case the primal violations for the constraints and variables are small meaning the solution is an almost feasible solution. Observe due to the rounding errors it can be expected that the violations are proportional to the size (`nrm:`) of the solution.
- Similarly for the dual solution the violations are small and hence the dual solution is almost feasible.
- Finally, it can be seen that the primal and dual objective values are almost identical.

To summarize in this case a primal and a dual solution only violate the primal and dual constraints slightly. Moreover, the primal and dual objective values are almost identical and hence it can be concluded that the reported solution is a good approximation to the optimal solution.

The reason the size (=norms) of the solution are shown is that it shows some about conditioning of the problem because if the primal and/or dual solution has very large norm then the violations and objective values are sensitive to small pertubations in the problem data. Therefore, the problem is unstable and care should be taken before using the solution.

Observe the function *task.solutionsummary* will print out the solution summary. In addition

- the problem status can be obtained using *task.getprosta*.
- the solution status can be obtained using *task.getsolsta*.
- the primal constraint and variable violations can be obtained with *task.getpviolcon* and *task.getpviolvar*.
- the dual constraint and variable violations can be obtained with *task.getdviolcon* and *task.getdviolvar* respectively.
- the primal and dual objective values can be obtained with *task.getprimalobj* and *task.getdualobj*.

Now what happens if the problem does not have an optimal solution e.g. is primal infeasible. In such a case the solution summary may look like

```
Interior-point solution summary
  Problem status  : PRIMAL_INFEASIBLE
  Solution status : PRIMAL_INFEASIBLE_CER
  Dual.    obj: 6.7319732555e+000   nrm: 8e+000   Viol.  con: 3e-010   var: 2e-009
```

i.e. **MOSEK** reports that the solution is a certificate of primal infeasibility but a certificate of primal infeasibility what does that mean? It means that the dual solution is a Farkas type certificate. Recall Farkas' Lemma says

$$\begin{array}{rcl} Ax & = & b, \\ x & \geq & 0 \end{array}$$

www.EngineeringBooksPdf.com

if and only if a y exists such that

$$\begin{array}{rcl} A^T y & \leq & 0, \\ b^T y & > & 0. \end{array} \tag{3.13}$$

Observe the infeasibility certificate has the same form as a regular dual solution and therefore the certificate is stored as a dual solution. In order to check quality of the primal infeasibility certificate it should be checked whether satisfies (3.13). Hence, the dual objective value is $b^T y$ should be strictly positive and the maximal violation in $A^T y \leq 0$ should be a small. In this case we conclude the certificate is of high quality because the dual objective is postive and large compared to the violations. Note the Farkas certificate is a ray so any postive multiple of that ray is also certificate. This implies the absolute of the value objective value and the violation is not relevant.

In the case a problem is dual infeasible then the solution summary may look like

```
Basic solution summary
Problem status  : DUAL_INFEASIBLE
Solution status : DUAL_INFEASIBLE_CER
Primal.  obj: -2.0000000000e-002  nrm: 1e+000   Viol.  con: 0e+000   var: 0e+000
```

Observe when a solution is a certificate of dual infeasibility then the primal solution contains the certificate. Moreoever, given the problem is a minimization problem the objective value should be negative and large compared to the worst violation if the certificate is strong.

Listing 3.13 shows how to use these function to determine the quality of the solution.

Listing 3.13: An example of solution quality analysis.

```
import sys
import mosek

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

if len(sys.argv) <= 1:
    print ("Missing argument, syntax is:")
    print ("  solutionquality inputfile")
else:

    try:

        # 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')
            task.readdata (sys.argv[1])

            # Solve the problem
            task.optimize ()

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

            whichsol= mosek.soltype.bas
```


www.EngineeringBooksPdf.com

```
            solsta= task.getsolsta(whichsol)

            ␣
↪pobj,pviolcon,pviolvar,pviolbarvar,pviolcones,pviolitg,dobj,dviolcon,dviolvar,dviolbarvar,dviolcones␣
↪= task.getsolutioninfo(whichsol)

            if solsta in [mosek.solsta.optimal,mosek.solsta.near_optimal]:

              abs_obj_gap     = abs(dobj-pobj)
              rel_obj_gap     = abs_obj_gap/(1.0 + min(abs(pobj),abs(dobj)))
              max_primal_viol = max(pviolcon,pviolvar)
              max_primal_viol = max(max_primal_viol  ,pviolbarvar)
              max_primal_viol = max(max_primal_viol  ,pviolcones)

              max_dual_viol   = max(dviolcon,dviolvar)
              max_dual_viol   = max(max_dual_viol  ,dviolbarvar)
              max_dual_viol   = max(max_dual_viol  ,dviolcones)

              # Assume the application needs the solution to be within
              #    1e-6 ofoptimality in an absolute sense. Another approach
              #   would be looking at the relative objective gap

              print ("\n\n")
              print ("Customized solution information.\n")
              print ("  Absolute objective gap: %e\n"%abs_obj_gap)
              print ("  Relative objective gap: %e\n"%rel_obj_gap)
              print ("  Max primal violation  : %e\n"%max_primal_viol)
              print ("  Max dual violation    : %e\n"%max_dual_viol)

              accepted= True

              if  rel_obj_gap>1e-6 :
                print ("Warning: The relative objective gap is LARGE.")
                accepted = False

              # We will accept a primal infeasibility of 1e-8 and
              # dual infeasibility of 1e-6. These number should chosen problem
              # dependent.
              if max_primal_viol>1e-8 :
                print ("Warning: Primal violation is too LARGE")
                accepted = False


              if max_dual_viol>1e-6 :
                print ("Warning: Dual violation is too LARGE.")
                accepted = False

              if accepted:

                numvar = task.getnumvar()
                print ("Optimal primal solution")
                xj=[0.]
                for j in range(numvar):
                    task.getxxslice(whichsol,j,j+1,xj)
                    print ("x[%d]: %e\n"%(j,xj[0]))

              else:
                #Print detailed information about the solution
                task.analyzesolution(mosek.streamtype.log,whichsol)

            elif solsta in [mosek.solsta.dual_infeas_cer, mosek.solsta.prim_infeas_cer,\
                            mosek.solsta.near_dual_infeas_cer, mosek.solsta.near_prim_infeas_
↪cer]:
```


www.EngineeringBooksPdf.com

```
            print ("Primal or dual infeasibility certificate found.")

          elif solsta == mosek.solsta.unkwown:
            print ("The status of the solution is unknown.")
          else:
            print ("Other solution status")


    except mosek.Error as e:
        print (e)
```

3.9.2 The Solution Summary for Mixed-Integer Problems

The solution summary for a mixed-integer problem may look like

Listing 3.14: Example of solution summary for a mixed-integer problem.

```
Integer solution solution summary
Problem status  : PRIMAL_FEASIBLE
Solution status : INTEGER_OPTIMAL
Primal.  obj: 3.4016000000e+005   nrm: 1e+000   Viol.  con: 0e+000   var: 0e+000   itg: 3e-014
```

The main diffrence compared to thecontinous case covered previously is that no information about the dual solution is provided. Simply because there is no dual solution available for a mixed integer problem. In this case it can be seen that the solution is highly feasible because the violations are small. Moreoever, the solution is denoted integer optimal. Observe *itg: 3e-014* implies that all the integer constrained variables are at most $3e-014$ from being an exact integer.

3.10 Solver Parameters

The **MOSEK** API provides many parameters to tune and customize the solver behaviour. Parameters are grouped depending on their type: integer, double or string. In general, it should not be necessary to change any of the parameters but if required, it is easily done. A complete list of all parameters is found in Section *16.4*.

We will show how to access and set the integer parameter that define the logging verbosity of the solver, i.e. *iparam.log*, and the algorithm used by **MOSEK**, i.e. *iparam.optimizer*.

Note: The very same concepts and procedures apply to string and double valued parameters.

To inspect the current value of a parameter, we can use the *task.getintparam*. In this example we say

```
param=  task.getintparam(mosek.iparam.log)
```

To set a parameter the **MOSEK** API provides several functions that differ in the way the parameter name and value are specified.

A parameter can be accessed by an identifier using *task.putintparam*

```
task.putintparam(mosek.iparam.log,1)
```

```
try:
    print ('setting to -1 using putintparam...')
    task.putintparam(mosek.iparam.log,-1)
```


www.EngineeringBooksPdf.com

```
except mosek.Error as e:
    print ('-1 rejected as not a valid value')
```

The values for integer parameters are either simple integer values or enum values. Enumerations are provided mainly to improve readability and ensure compatibility.

In the next lines we show how to set the algorithm used by **MOSEK** to solve linear optimization problem. To that purpose we set the *iparam.optimizer* parameter using a value from the *optimizertype* enumeration: for instance we may decide to use the dual simplex algorithm, and thus

```
task.putintparam(mosek.iparam.optimizer,mosek.optimizertype.dual_simplex)
```

For more information about other parameter related functions, please browse the API reference in Section *16*.

The complete code for this tutorial follows in Listing 3.15.

Listing 3.15: Parameter setting example.

```
import mosek

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

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

    print ('Test MOSEK parameter get/set functions')

    param=  task.getintparam(mosek.iparam.log)
    print ('default value for parameter mosek.ipar.log= ',param)

    print (' setting to 1 using putintparam...')
    task.putintparam(mosek.iparam.log,1)


    try:
        print ('setting to -1 using putintparam...')
        task.putintparam(mosek.iparam.log,-1)
    except mosek.Error as e:
        print ('-1 rejected as not a valid value')

    print ('setting to 2 using putparam...')
    task.putparam('MSK_IPAR_LOG','2')

    print ('setting to 3 using putnaintparam...')
    task.putnaintparam('MSK_IPAR_LOG',3)

    print ('selecting the dual simplex algorithm...')
    task.putintparam(mosek.iparam.optimizer,mosek.optimizertype.dual_simplex)
```


www.EngineeringBooksPdf.com

www.EngineeringBooksPdf.com

CHAPTER

FOUR

NONLINEAR TUTORIALS

This chapter provides information about how to solve general convex nonlinear optimization problems using **MOSEK**. By general nonlinear problems it is meant problems that cannot be formulated as a conic quadratic optimization or a convex quadratically constrained optimization problem.

In general it is recommended not to use nonlinear optimizer unless needed. The reasons are

- **MOSEK** has no way of checking whether the formulated problem is convex and if this assumption is not satisfied the optimizer will not work.
- The nonlinear optimizer requires 1st and 2nd order derivative information which is hard to provide correctly i.e. it is nontrivial to program the code that computes the derivative information.
- The algorithm employed for nonlinear optimization problems is not as good as the one employed for conic problems i.e. conic problems has special that can be exploited to make the optimizer faster and more robust.

This leads to following advices in decreasing order of importance.

1. Consider reformulating the problem to a conic quadratic optimization problem if at all possible. In particular many problems involving polynomial terms can easily be reformulated to conic quadratic form.
2. Consider reformulating the problem to a separable optimization problem because that simplifies the issue with verifying convexity and computing 1st and 2nd order derivatives significantly. In most cases problems on separable form also solves faster because of the simpler structure of the functions.
3. Finally, if the problem cannot be reformulated to separable form then use a modelling language like AMPL or GAMS. The reason is the modeling language will do all the computing of function values and derivatives. This eliminates an important source of errors. Therefore, it is strongly recommended to use a modelling language at the prototype stage.

4.1 Separable Convex (SCopt) Interface

The **MOSEK** optimizer API provides a way to add simple non-linear functions composed from a limited set of non-linear terms. Non-linear terms can be mixed with quadratic terms in objective and constraints.

We consider a normal linear problem with additional non-linear terms z:

$$\begin{array}{lccccc} \text{minimize} & & & z_0(x) + c^T x & & \\ \text{subject to} & l_i^c & \leq & z_i(x) + a_i^T x & \leq & u_i^c,\ i = 1 \ldots m \\ & l^x & \leq & x & \leq & u^x, \\ & x \in \mathbb{R}^n & z : \mathbb{R}^n \to \mathbb{R}^{(m+1)} & & & \end{array}$$

Using the separable non-linear interface it is possible to add non-linear functions of the form

$$z_i(x) = \sum_{k=1}^{K_i} w_k^i(x_{p_{ik}}),\ w_k^i : \mathbb{R} \to \mathbb{R}$$

www.EngineeringBooksPdf.com

In other words, each non-linear function z_i is a sum of separable functions w_k^i of one variable each. A limited set of functions are supported; each w_k^i can be one of the separable functions:

Table 4.1: Functions supported by the SCopt interface.

Separable function	Operator name	
$fx\ln(x)$	*ent*	Entropy function
fe^{gx+h}	*exp*	Exponential function
$f\ln(gx+h)$	*log*	Logarithm
$f(x+h)^g$	*pow*	Power function

where f, g and h are constants.

This formulation does not guarantee convexity. For **MOSEK** to be able to solve the problem, following requirements must be met:

- If the objective is minimized, the sum of non-linear terms must be convex, otherwise it must be concave.
- Any constraint bounded below must be concave, and any constraint bounded above must be convex.
- Each separable term must be twice differentiable within the bounds of the variable it is applied to.

If these are not satisfied **MOSEK** may not be able to solve the problem or produce a meaningful status report. For details see Section *4.1.1*.

Important: When to use the SCopt API:

- When conic can absolutely not be used.
- when a conic formulation would be significantly larger

Problems: less stable, less predictable, harder to debug, worse status info

---

4.1.1 Ensuring Convexity and Differentiability

Some simple rules can be set up to ensure that the problem satisfies **MOSEK**'s convexity and differentiability requirements. First of all, for any variable x_i used in a separable term, the variable bounds must define a range within which the function is twice differentiable.

We can define these bounds as follows:

Separable function	Operator name	Safe x bounds
$fx\ln(x)$	*ent*	$0 < x$.
fe^{gx+h}	*exp*	$-\infty < x < \infty$.
$f\ln(gx+h)$	*log*	If $g > 0$: $-h/g < x$.
		If $g < 0$: $x < -h/g$.
$f(x+h)^g$	*pow*	If $g > 0$ and integer: $-\infty < x < \infty$.
		If $g < 0$ and integer: either $-h < x$ or $x < -h$.
		Otherwise: $-h < x$.

To ensure convexity, we require that each $z_i(x)$ is either a sum of convex terms or a sum of concave terms. The following table lists convexity for the relevant ranges for $f > 0$ — changing the sign of f switches concavity/convexity.

www.EngineeringBooksPdf.com

Separable function	Operator name	Convexity conditions
$fx\ln(x)$	*ent*	Convex within safe bounds.
fe^{gx+h}	*exp*	Convex for all x.
$f\ln(gx+h)$	*log*	Concave within safe bounds.
$f(x+h)^g$	*pow*	If g is even integer: convex within safe bounds.
		If g is odd integer: • concave if $(-\infty, -h)$, • convex if $(-h, \infty)$ If $0 < g < 1$: concave within safe bounds.
		Otherwise: convex within safe bounds.

4.1.2 SCopt Example

Subsequently, we will use the following example to demonstrate the solution of a separable convex optimization problem using **MOSEK**

$$\begin{array}{ll} \text{maximize} & \exp(x_2) - \ln(x_1) \\ \text{subject to} & x_2\ln(x_2) \geq 0 \\ & x_1^{\sqrt{2}} - x_2 \leq 0 \\ & x_1, x_2 \geq \frac{1}{2}. \end{array} \tag{4.1}$$

This problem is obviously separable. Moreover, note that all nonlinear functions are well defined for x values satisfying the variable bounds strictly, i.e.

$$x_1, x_2 > 0.$$

This assures that function evaluation errors will not occur during the optimization process because **MOSEK** will only evaluate $\ln(x_1)$ and $x_2\ln(x_2)$ for $x_1, x_2 > 0$.

The method employed above can often be used to make convex optimization problems separable even if these are not formulated as such initially. The reader might object that this approach is inefficient because additional constraints and variables are introduced to make the problem separable. However, in our experience this drawback is offset largely by the much simpler structure of the nonlinear functions. Particularly, the evaluation of the nonlinear functions, their gradients and Hessians is much easier in the separable case.

The complete source code follows in Listing 4.1.

Listing 4.1: Implementation of problem (4.1).

```
import sys

import mosek

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

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

      numvar = 2
      numcon = 2
```


www.EngineeringBooksPdf.com

```
      inf = 0.

      bkc = [ mosek.boundkey.up,
              mosek.boundkey.lo]
      blc = [  -inf,  0.]
      buc = [    0., inf]

      bkx = [ mosek.boundkey.ra] * numvar
      blx = [ 0.5 ] * numvar
      bux = [ 1.0 ] * numvar

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

      task.putvarboundslice(0, numvar, bkx, blx, bux)
      task.putconboundslice(0, numcon, bkc, blc, buc)

      task.putaij(1, 1, -1.0)

      opro  = [ mosek.scopr.log, mosek.scopr.exp ]
      oprjo = [               0,               1 ]
      oprfo = [            -1.0,             1.0 ]
      oprgo = [             1.0,             1.0 ]
      oprho = [             0.0,             0.0 ]


      oprc  = [ mosek.scopr.ent, mosek.scopr.pow ]
      opric = [               0,               1 ]
      oprjc = [               1,               0 ]
      oprfc = [             1.0,             1.0 ]
      oprgc = [             0.0,             0.5 ]
      oprhc = [             0.0,             0.0 ]

      task.putSCeval(opro, oprjo, oprfo, oprgo, oprho,
                     oprc, opric, oprjc, oprfc, oprgc, oprhc)

      task.optimize()

      res = [ 0.0 ] * numvar
      task.getsolutionslice(
        mosek.soltype.itr,
        mosek.solitem.xx,
        0, numvar,
        res)

      print ( "Solution is: %s" % res )
      task.putintparam(mosek.iparam.write_ignore_incompatible_items,mosek.onoffkey.on)
      task.writeSC("scprob.sc","scprob.opf")

main()
```


www.EngineeringBooksPdf.com

CHAPTER

FIVE

ADVANCED TUTORIALS

5.1 The Progress Call-back

Some of the API function calls, notably *task.optimize*, may take a long time to complete. Therefore, during the optimization a call-back function is called frequently, to provide information on the progress of the call. From the call-back function it is possible

- to obtain information on the solution process,
- to report of the optimizer's progress, and
- to ask **MOSEK** to terminate, if desired.

The call-back function arguments provide the following information:

- a code that identify the event that caused the call-back to be called,
- an handle to a user-defined data structure and
- the complete set of parameters used by the solver.

The user can force the solver to stop using the return value of the call-back.

Listing 5.1 shows how the progress call-back function can be used.

Listing 5.1: Progress call back example

```
import sys

import mosek
from mosek import *


def makeUserCallback(maxtime):
    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))
```


www.EngineeringBooksPdf.com

```
        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.begin_bi:
            print ("Basis identification started.")
        elif caller == callbackcode.end_bi:
            print ("Basis identification finished.")
        else:
            pass

        if opttime >= maxtime:
            # mosek is spending too much time. Terminate it.
            return 1

        return 0
    return userCallback

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

def main(args):
  filename = "../data/25fv47.mps"
  slvr     = "intpnt"

  if len(args) < 3:
      print("Usage: callback ( psim | dsim | intpnt ) filename")

  if len(args) > 1: slvr = args[1]
  if len(args) > 2: filename = args[2]


  with mosek.Env() as env:
      with mosek.Task(env) as task:
          task.readdata(filename)

          task.set_Stream(streamtype.log, msgPrinter)

          if   slvr == 'psim':
```


www.EngineeringBooksPdf.com

```
            task.putintparam(iparam.optimizer,optimizertype.primal_simplex)
        elif slvr == "dsim":
            task.putintparam(iparam.optimizer,optimizertype.dual_simplex)
        elif slvr == "intpnt":
            task.putintparam(iparam.optimizer,optimizertype.intpnt)

        # Turn all MOSEK logging off (note that errors and other messages
        # are still sent through the log stream)
        task.putintparam(iparam.log, 0)

        usercallback = makeUserCallback(maxtime = 3600)
        task.set_Progress(usercallback)

        task.optimize()

        task.solutionsummary(streamtype.msg)

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

Important: Due to the way the Python garbage collector works, it is necessary to hold a reference to the callback function for the duration of the lifetime of the *Task* object. This means that a construction such as

```
#NOTE: WRONG way to attach callback!
task.set_Progress(lambda caller, dinf, iinf, liinf: print "Caller : %d" % caller)
```

will **not** work.

5.2 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 that B^{-1} exists. This implies that the linear systems

$$B\bar{x} = w \tag{5.1}$$

and

$$B^T\bar{x} = w \tag{5.2}$$

each has a unique solution for all w .

MOSEK provides functions for solving the linear systems (5.1) and (5.2) for an arbitrary w.

In the next sections we will show how to use **MOSEK** to

- *identify the solution basis*,
- *solve arbitrary linear systems*.

www.EngineeringBooksPdf.com

5.2.1 Identifying the Basis

To use the solutions to (5.1) and (5.2) it is important to know how the basis matrix B is constructed.

Internally **MOSEK** employs the linear optimization problem

$$\begin{array}{lrcl} \text{maximize} & & c^T x & \\ \text{subject to} & & Ax - x^c & = 0, \\ & l^x \leq & x & \leq u^x, \\ & l^c \leq & x^c & \leq u^c. \end{array} \tag{5.3}$$

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(basis)
```

where `basis` is an array of variable indexes.

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

$$\texttt{basis}[i] < \texttt{numcon},$$

then the i'th basis variable is x_i^c. 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 variable

$$x_{\texttt{basis}[i]-\texttt{numcon}}$$

and the i 'th column of B is the column

$$A_{:,(\texttt{basis}[i]-\texttt{numcon})}.$$

For instance if $\texttt{basis}[0] = 4$ and $\texttt{numcon} = 5$, then since $\texttt{basis}[0] < \texttt{numcon}$, the first basis variable is x_4^c. Therefore, the first column of B is the fourth column of $-I$. Similarly, if $\texttt{basis}[1] = 7$, then the second variable in the basis is $x_{\texttt{basis}[1]-\texttt{numcon}} = x_2$. Hence, the second column of B is identical to $a_{:,2}$.

An example

Consider the linear optimization problem:

$$\begin{array}{lrcl} \text{minimize} & x_0 + x_1 & & \\ \text{subject to} & x_0 + 2x_1 & \leq & 2, \\ & x_0 + x_1 & \leq & 6, \\ & x_0, x_1 \geq 0. & & \end{array} \tag{5.4}$$

Suppose a call to *task.initbasissolve* returns an array `basis` so that

www.EngineeringBooksPdf.com

```
basis[0] = 1,
basis[1] = 2.
```

Then the basis variables are x_1^c and x_0 and the corresponding basis matrix B is

$$\begin{bmatrix} 0 & 1 \\ -1 & 1 \end{bmatrix}.$$

Please note the ordering of the columns in B .

The program in Listing 5.2 demonstrates the use of *task.solvewithbasis*.

Listing 5.2: A program showing how to identify the basis.

```
import mosek

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

def main():
    numcon = 2
    numvar = 2

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

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

    blc  = [-infinity,
            -infinity]
    buc  = [2.0,
            6.0]

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

    bux  = [+infinity,
            +infinity]
    w1 = [2.0, 6.0]
    w2 = [1.0, 0.0]
    try:
      with mosek.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,
```


www.EngineeringBooksPdf.com

```
                          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]))
            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 (5.4) 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.

www.EngineeringBooksPdf.com

```
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

$$\begin{bmatrix} 0 & 1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} x_1^c \\ x_0 \end{bmatrix} = \begin{bmatrix} 2 \\ 6 \end{bmatrix}.$$

5.2.2 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.

Below we demonstrate how to solve the linear system

$$\begin{bmatrix} 0 & 1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \tag{5.5}$$

with $b = (1, -2)$ and $b = (7, 0)$.

```
import mosek

def put_a( task,
           aval,
           asub,
           ptrb,
           ptre,
           numvar,
           basis):
```


www.EngineeringBooksPdf.com

```
    # 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):
      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.

    for i in range(numvar):
      task.putsolutioni (mosek.accmode.var,
                         i,
                         mosek.soltype.bas,
                         skx[i],
                         0.0,
                         0.0,
                         0.0,
                         0.0)

    for i in range(numvar):
      task.putsolutioni (mosek.accmode.con,
                         i,
                         mosek.soltype.bas,
                         skc[i],
                         0.0,
                         0.0,
                         0.0,
                         0.0)

    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];
```


www.EngineeringBooksPdf.com

```
    #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 ]

        put_a(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")

        # 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 using the *task.putsolutioni* function, where we define the status key for each variable. The actual values

www.EngineeringBooksPdf.com

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
```

and we can verify that $x_0 = 2, x_1 = -4$ is indeed a solution to (5.5).

5.3 Calling BLAS/LAPACK Routines from MOSEK

Sometimes users need to perform linear algebra operations that involve dense matrices and vectors. Also **MOSEK** uses extensively high-performance linear algebra routines from the BLAS and LAPACK packages and some of this routine are included in the package shipped to the users.

MOSEK makes available to the user some BLAS and LAPACK routines by **MOSEK** functions that

- use **MOSEK** data types and response code;
- keep BLAS/LAPACK naming convention.

Therefore the user can leverage on efficient linear algebra routines, with a simplified interface, with no need for additional packages. In the Table 5.1 we list BLAS functions.

Table 5.1: 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*	$C = \alpha AA^T + \beta C$

Function from LAPACK are listed in Table 5.2.

Table 5.2: LAPACK functions available from **MOSEK**

LAPACK Name	**MOSEK** function	Description
POTRF	*env.potrf*	Cholesky factorization of a semidefinite symmetric matrix
SYEVD	*env.syevd*	Eigen-values of a symmetric matrix
SYEIG	*env.syeig*	Eigen-values and eigen-vectors of a symmetric matrix

Click on the **MOSEK** function link to access additional information.

A working example

In Listing 5.3 we provide a simple working example. It has no practical meaning except to show how to call the provided functions and how the input should be organized.

www.EngineeringBooksPdf.com

Listing 5.3: A working example on how to call BLAS and LAPACK routines from **MOSEK**.

```
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]

#  routines

    xy = env.dot(n,x,y)
    print("dot results= %f\n"%xy);

    env.axpy(n,alpha,x,y)
    print("\naxpy results is ")
    print_matrix(y,1,len(y))

    env.gemv(mosek.transpose.no, m, n, alpha, A, x, beta,z)
    print("\ngemv results is ")
    print_matrix(z,1,len(z))

    env.gemm(mosek.transpose.no,mosek.transpose.no,m,n,k,alpha,A,B,beta,C)
    print("\ngemm results is ")
    print_matrix(C,m,n)

    env.syrk(mosek.uplo.lo, mosek.transpose.no,m,k,alpha, A, beta,D)
    print("\nsyrk results is")
    print_matrix(D,m,m)

# LAPACK routines

    env.potrf(mosek.uplo.lo,m,Q)
    print("\npotrf results is ")
    print_matrix(Q,m,m)


    env.syeig(mosek.uplo.lo,m,Q,v)
    print("\nsyeig results is")
```


www.EngineeringBooksPdf.com

```
    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...")
```

5.4 Computing a Sparse Cholesky Factorization

Given a positive semidefinite symmetric matrix

$$A \in \mathbb{R}^{n \times n}$$

then it is well known there exists a matrix L such that

$$A = LL^T.$$

If the matrix L is a lower triagular matrix then it is called a Cholesky factorization.

Given A is positive definite i.e. nonsingular then L is also nonsingular. This implies that the linear equation system

$$Ax = b$$

can be solved by first solving the lower triangular system

$$Ly = b$$

and then the upper triangular system

$$L^T x = y.$$

For this reason only a Cholesky factorization is useful. Therefore, **MOSEK** provides function that can compute a Cholesky factorization of a positive semidefinite matrix. In addition a function for performing solves with a nonsingular lower triangular matrix and its tranpose is availble.

In practice A may be very large e.g. n is in the range of millions. However, then A is typically sparse which means that most of the elements in A are zero. Fortunately the sparsity can be exploited during computations of the Cholesky factorization to reduce the computational cost. How large the computational savings are is dependent of the position of the zeros. This can be demonstrated using the example

$$A = \begin{bmatrix} 4 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}. \tag{5.6}$$

Now if we compute a Cholesky factorization with 5 digits of accuracy we obtain

$$\begin{bmatrix} 2.0000 & 0 & 0 & 0 \\ 0.5000 & 0.8660 & 0 & 0 \\ 0.5000 & -0.2887 & 0.8165 & 0 \\ 0.5000 & -0.2887 & -0.4082 & 0.7071 \end{bmatrix}.$$

www.EngineeringBooksPdf.com

Next let us permute the rows and columns symmetrically of A i.e. we multiply A by the permutation matrix

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

as follows

$$PAP^T = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 4 \end{bmatrix}.$$

Now the Cholesky factorization of

$$PAP^T$$

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 of the original matrix.

Determining a permutation matrix that leads to the sparsest Cholesky factorization or the minimal amount of work is NP hard. Hence the choice of a good permutation can only be determined heuristically using for instance the minimum degree heuristic and variants. The function provided by **MOSEK** for computing a Cholesky factorization has build in a permutation aka. reordering heuristic.

Next let us see how the Cholesky factorization of (5.6) can be computed using **MOSEK**. The complete source code is in Listing 5.4.

Listing 5.4: How to use the sparse Cholesky factorization routine available in **MOSEK**.

```
import sys
import mosek

def sparsecholesky(env,
                   n,
                   anzc,
                   asubc,
                   aptrc,
                   avalc):

  multiworker = 0         # Diasble multithreading since the problems are small.
  order_meth  = 1         # Permute/reorder to save computational work
  tolsingular = 1.0e-14

  try:
    perm,diag,lnzc,lptrc,lensubnval,lsubc,lvalc = env.computesparsecholesky(multiworker,
                                                                            order_meth,
                                                                            tolsingular,
                                                                            anzc,
                                                                            aptrc,
                                                                            asubc,
                                                                            avalc)

    print("L and D. Length=%d\n"% lensubnval);
```


www.EngineeringBooksPdf.com

```
    for i in range(n):

      print ( ' '.join( [ "%d perm=%d diag=%.4e :" % (i, perm[i], diag[i])] +
                        [ " %.4e[%d]" % (lvalc[lptrc[i]+s],lsubc[lptrc[i]+s]) for s in␣
↪range(lnzc[i]) ]
                      )
            )

  except:
    raise

  return perm,diag,lnzc,lptrc,lensubnval,lsubc,lvalc


print("Sparse Cholesky computation.")

# Create the mosek environment.
with mosek.Env() as env:
    with env.Task(0,0) as task:

      # Let A be
      #
      #     [4.0  1.0 1.0 1.0]
      #     [1.0  1.0       ]
      #     [1.0      1.0   ]
      #     [1.0          1.0]
      #
      # then
      #
      #    a. Compute a sparse Cholesky factorization A.
      #    b. Solve the linear system A x = b using the Cholesky factor
      #
      # Observe that anzc, aptrc, asubc and avalc only specify the lower triangular part.

      n     = 4
      anzc  = [4, 1, 1, 1]
      asubc = [0, 1, 2, 3, 1, 2, 3]
      aptrc = [0, 4, 5, 6]
      avalc = [4.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
      b     = [13.0, 3.0, 4.0, 5.0]



      print("First example with a definite A.")


      perm, diag, lnzc, lptrc, lensubnval, lsubc, lvalc = sparsecholesky(env,
                                                                         n,
                                                                         anzc,
                                                                         asubc,
                                                                         aptrc,
                                                                         avalc)


      # Permuted b is stored as x.
      x = [ b[p] for p in perm ]

      # Compute  x = inv(L)*x.
      env.sparsetriangularsolvedense(mosek.transpose.no,
                                     lnzc,
                                     lptrc,
                                     lsubc,
                                     lvalc,
```


www.EngineeringBooksPdf.com

```
                                    x)


    # Compute  x = inv(L^T)*x.
    env.sparsetriangularsolvedense(mosek.transpose.yes,
                                    lnzc,
                                    lptrc,
                                    lsubc,
                                    lvalc,
                                    x)


    print("\n\nSolution A x = b")
    print('\n'.join([ "x[%d]: %.2e"% (pp,xx) for pp,xx in zip(perm,x) ] ) )

    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]

    # Let A be
    #
    # [1.0 1.0 1.0]
    # [1.0 1.0 1.0]
    # [1.0 1.0 1.0]
    #
    # then compute a sparse Cholesky factorization A. Observe A is NOT
    # positive definite.
    #


    print("Second example with a semidefinite A.")

    perm, diag, lnzc, lptrc, lensubnval, lsubc, lvalc = sparsecholesky(env,
                                                                       n,
                                                                       anzc,
                                                                       asubc,
                                                                       aptrc,
                                                                       avalc)
```

Here is the output when running the code

```
Sparse Cholesky computation.

First example with a definite A.
L and D. Length=7
0 perm=3 diag=0.0000e+000 : 1.0000e+000[0] 1.0000e+000[2]
1 perm=2 diag=0.0000e+000 : 1.0000e+000[1] 1.0000e+000[2]
2 perm=0 diag=0.0000e+000 : 1.4142e+000[2] 7.0711e-001[3]
3 perm=1 diag=0.0000e+000 : 7.0711e-001[3]

Solution A x = b
x[3]: 4.00e+000
x[2]: 3.00e+000
x[0]: 1.00e+000
x[1]: 2.00e+000

Second example with a semidefinite A.
L and D. Length=6
0 perm=0 diag=0.0000e+000 : 1.0000e+000[0] 1.0000e+000[1] 1.0000e+000[2]
1 perm=2 diag=1.0000e-014 : 1.0000e-007[1] 0.0000e+000[2]
```


www.EngineeringBooksPdf.com

```
2 perm=1 diag=1.0000e-014 : 1.0000e-007[2]
```

From the output the perm array has the value

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

implying the permutation matrix

$$P = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

has been employed. Now the Cholesky factorization of the permuted matrix is reported to be

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1.4142 & 0 \\ 0 & 0 & 0.7071 & 0.7071 \end{bmatrix}$$

using 5 figures of accuracy which can be verified to be correct.

After computing the Cholesky factorization has been computed it is used to solve the linear equation system

$$Ax = b$$

where b is

$$\begin{bmatrix} 13.0 & 3.0 & 4.0 & 5.0 \end{bmatrix}^T.$$

The solutiuon is reported to be

$$\begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix}^T$$

which is correct.

The second example in the source code shows what happens if a sparse Cholesky factorization of a semidefinite matrix is computed. The example A is

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}^T \tag{5.7}$$

which is a rank 1 matrix. The results is given by

$$PAP^T + D = LL^T$$

where

$$D = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1.0e-14 & 0 \\ 0 & 0 & 1.0e-14 \end{bmatrix},$$

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1.0e-7 & 0 \\ 1 & 0 & 1.0e-7 \end{bmatrix}$$

and

$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}.$$

www.EngineeringBooksPdf.com

Since A is only positive semdefinite i.e. not of full rank then some of diagonal elements of A are boosted to make it truely positive definite. The amount of boosting is given as an argument to the function computing the sparse Cholesky factorization and has been chosen to 1.0e-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 which in general is the best we hope for given computations are performed finite precision.

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.

5.5 Converting a quadratically constrained problem to conic form

A conic quadratic constraint has the form

$$x \in \mathcal{Q}^n$$

in its most basic form where

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

Alternatively the conic constraint can be represented using a quadratic inequality

$$\sum_{j=2}^{n} x_j^2 - x_1^2 \leq 0.0 \tag{5.8}$$

and the simple linear inequality

$$x_1 \geq 0.0.$$

Therefore, it is possible to state conic quadratic problems using quadratic inequalities. Some drawbacks of specifying conic quadratic problems using quadratic inequalities are

- the elegant duality theory for conic problems is lost.
- reporting accurate dual information for quadratic inequalities is hard and/or compuational expensive.
- the left hand side of (5.8) is nonconvex so the formulation is stricly speaking not convex.
- a computational overhead is introduced when converting the quadratic inequalities to conic form before optimizing.
- modelling directly on conic form usually leads to a better model *[And13]* i.e. a faster solution time and better numerical properties.

In addition quadratic inequalities can not be used to speciffy the semidefinite cone or other more general cones than quadratic cone. Despite the drawbacks it is not uncommon to state conic quadratic problems using quadratic inequalities and therefore **MOSEK** has a function that translate certain quadratically constrained problems to conic form. Note that the **MOSEK** interior-point optimizer will do that automatically for convex quadratic problems automatically. So quadratic to conic form conversion is primarily useful for problems having conic quadratic constraints embedded.

MOSEK employs the following form of quadratic problems:

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

www.EngineeringBooksPdf.com

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,
- for each reformulated quadratic constraint there will be:
  - one rotated quadratic cone for each quadratic constraint,
  - one rotated quadratic cone if the objective function is quadratic,
  - 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.

5.5.1 Quadratic Constraint Reformulation

Let us assume we want to convert the following quadratic constraint

$$l \leq \frac{1}{2} x^T Q x + \sum_{j=0}^{n-1} a_j x_j \leq u$$

in conic form. We first check it must hold either $l = -\infty$ or $u = \infty$, otherwise either the constraint can be dropped, or the constraint is not convex. Thus let us consider the case

$$\frac{1}{2} x^T Q x + \sum_{j=0}^{n-1} a_j^T x_j \leq u. \tag{5.9}$$

Introducing an additional variable w such that

$$w = u - \sum_{j=0}^{n-1} a_j^T x_j \tag{5.10}$$

we obtain the equivalent form

$$\begin{array}{lcl} \frac{1}{2} x^T Q x & \leq & w, \\ u - \sum_{j=0}^{n-1} a_j x_j & = & w. \end{array}$$

If Q is positive semidefinite, then there exists a matrix F such that

$$Q = F F^T \tag{5.11}$$

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

$$\begin{array}{ll} (w, z, y) & \in \mathcal{Q}_r, \\ z = 1. & \\ y = Fx & \\ w = u - \sum_{j=0}^{n-1} a_j^T x_j. & \end{array} \tag{5.12}$$

Summarizing, for each quadratic constraint involving $t \leq n$ variables, **MOSEK** introduces

www.EngineeringBooksPdf.com

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.

5.5.2 Some Examples

We report in this section few examples of reformulation of a QCQP problem in conic form. For each problem we will show its definition before and after the reformulation, using the human-readable *OPF format*.

Quadratic problem

We consider a simple quadratic problem of the form

$$\begin{array}{ll} \min & \frac{1}{2}(13x_0^2 + 17x_1^2 + 12x_2^2 + 24x_0x_1 + 12x_1x_2 - 4x_0x_2) - 22x_0 - 14.5x_1 + 12x_2 + 1 \\ s.t. & \\ & -1 \leq x_i \leq 1 \qquad\qquad i = 0, 1, 2 \end{array}$$

```
[comment]
An example of small QP from Boyd and Vandenberghe, "Convex Optimization", pag 189 ex 4.3
The solution is (1,0.5,-1)
[/comment]


[variables disallow_new_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 solution 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:

$$\begin{array}{ll} \min & -22x0 - 14.5ex1 + 12x2 + x_3 + 1 \\ s.t. & \\ & 3.61x_0 + 3.33x_1 - 0.55x_2 - x_6 = 0 \\ & +2.29x_1 3.42x_2 - x_7 = 0 \\ & 0.81x_1 - x_8 = 0 \\ & -x_3 + x_4 = 0 \\ & x_5 = 1 \\ & (x_4, x_5, x_6, x_7, x_8) \in \mathcal{Q}_\nabla \\ & -1 \leq x_0, x_1, x_2 \leq 1 \end{array}$$

The model generated by *task.toconic* is

```
[comment]
Written by MOSEK version 8.0.0.8
Date 25-05-15
```


www.EngineeringBooksPdf.com

```
Time 15:51:41
[/comment]


[variables disallow_new_variables]
x0000_x0 x0001_x1 x0002_x2 x0003 x0004
x0005 x0006 x0007 x0008
[/variables]

[objective minimize]
- 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]          0 <= * [/b]
[b] -1e+00     <= x0000_x0,x0001_x1,x0002_x2 <= 1e+00 [/b]
[b]               x0003 free [/b]
[b]               x0005 =  1e+00 [/b]
[b]               x0006,x0007,x0008 free [/b]
[cone rquad k0000] x0005, x0004, x0006, x0007, x0008 [/cone]
[/bounds]
```

We can clearly see that constraints *c0000* to *c0002* represent the linear mapping as in (5.11), while (5.10) corresponds to *c0003*. The cone roots are *x0005* and *x0004*.

5.6 MOSEK OptServer

MOSEK provides an easy way to offload optimization problem to a remote server in both *synchronous* or *asynchronous* mode. This section describes the functionalities from the client side, i.e. how a user can *send* a given optimization problem to a remote server where a optimization server is listening and will run **MOSEK** to solve the problem.

5.6.1 Synchronous Remote Optimization

In synchoronous mdoe the client send the optimization problem to the optimization server and wait for the optimization to end. Once the result has been received, the program can continue. This is the simplest mode and requires very limited modifications to existing code: instead of *task.optimize* we only need to use *task.optimizermt* instead, passing the host and port on which the server is running and listening.

The rest of the code remains untouched.

Important: There is no way to recover a job in case the connection has been broken!

In Listing 5.5 we show how to modify tutorial in Section *3.1* so that the computation is off loaded to a remote machine.

www.EngineeringBooksPdf.com

Listing 5.5: 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')
            task.readdata (inputfile)

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

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

5.6.2 Asynchronous Remote Optimization

Working in asynchronous mode involves more steps. In particular once that the optimization has started, the user is responsible to check the status and when optimization ends, fetch the results. The user can also stop the optimization anytime. The relevant functions are:

- *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.

In Listing 5.6 the code in Listing 5.5 is extended in order to run asynchronously: after that the optimization is started, the program enters in a polling loop that regularly checks whether the result of the optimization is available.

Listing 5.6: Using the OptServer in synchronous mode.

```
import mosek
import sys
import time
```


www.EngineeringBooksPdf.com

```
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: ",token)

        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, trm, res = task.asyncpoll( host,
                                                          port,
                                                          token)


                print("done!")

                if respavailable:
                    print("solution available!")
                    respavailable, trm, res = 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)
```


www.EngineeringBooksPdf.com

www.EngineeringBooksPdf.com

www.EngineeringBooksPdf.com

CHAPTER

SIX

GUIDELINES

6.1 Deployment

When redistributing a Python application using the **MOSEK** Optimizer API for Python 8.0.0.64, the following libraries must be included:

64-bit Linux	64-bit Windows	32-bit Windows	64-bit Mac OS
`libmosek64.so.8.0`	`mosek64_8_0.dll`	`mosek32_8_0.dll`	`libmosek64.dylib.8.0`
`libiomp5.lib`	`libiomp5md.dll`	`libomp5md.dll`	`libiomp5.dylib`
`libcilkrts.so.5`	`cilkrts20.dll`	`cilkrts20.dll`	`libcilkrts.5.dylib`
`libmosekxx8_0.so`	`mosekxx8_0.dll`	`mosekxx8_0.dll`	`libmosekxx8_0.dylib`
`libmosekscopt8_0.so`	`mosekscopt8_0.dll`	`mosekscopt8_0.dll`	`libmosekscopt8_0.dylib`

Furthermore, one (or both) of the directories

- `python/2/mosek` for Python 2.x applications, and
- `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`. Alternative, if the binary libraries reside in another directory, the application can pre-load the `mosekxx` library from another located before `mosek` is imported, e.g. like this

```
import ctypes ; ctypes.CDLL('my/path/to/mosekxx.dll')
```

6.2 Efficiency Considerations

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

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.

www.EngineeringBooksPdf.com

- *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 change the problem, they only give hints to the eventual dimension of the problem. If the problem ends up growing larger than this, the estimates are automatically increased.

Do not mix `put-` and `get-` functions

For instance, the functions *task.putacol* and *task.getacol*. **MOSEK** will queue `put-` commands internally until a `get-` function is called. If every `put-` function call is followed by a `get-` function call, the queue will have to be flushed often, decreasing efficiency.

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

Use the LIFO principle

When removing constraints and variables, try to use a LIFO approach, i.e. Last In First out. **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 doing 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.

Use one environment (env) only

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

6.2.1 API overhead

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

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

The following rules will often improve the performance of the **MOSEK**/ Python API:

www.EngineeringBooksPdf.com

- Reuse `Env` and `Task` whenever possible : There may be some overhead involved in creating and deleting task and environment objects, so if possible reuse these.
- Make sure to delete task and environment when not in use anymore. Using the `with`-construction (available in Python 2.6 and later) will allow automatic deletion of the environment and task. If this is not an option, use `Env.__del__()` and `Task.__del__()` to destroy the objects. Failing to do this may cause memory leaks in some cases.
- Avoid input loops: Whenever possible imput data in large chunks or vectors instead of using loops. For small `put-` and `get-` methods there is a significant overhead, so for example inputting one row of the A-matrix at the time may be much slower than inputting the whole matrix.

For example, a loop with *task.putarow* may be replaced with one *task.putarowlist*, or a loop of *task.putqobjij* may be replaced with *task.putqobj*.

6.3 The license system

MOSEK is a commercial product that **always** needs a valid license to work. A license is typically provided as a license file that allows the user to access the subset of the **MOSEK** Optimization Suite functionalities it is entitled for, and for the right amount of time. **MOSEK** uses a third party license manager to implement license checking.

By default a license token remains checked out for the duration of the **MOSEK** session, i.e.

1. a license token is checked out when *task.optimize* is first called and
2. 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.

To change the license systems behavior to returning the license token after each call to **MOSEK** set the parameter *iparam.cache_license* to *onoffkey.off*.

Additionally license checkout and checkin can be controlled manually with the functions *env.checkinlicense* and *env.checkoutlicense*.

6.3.1 Waiting for a free license

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

6.3.2 Manually stopping the license system

Usually the license system is stopped automatically when the **MOSEK** library is unloaded. However, when the user explicitly unload the library using e.g windows 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**.

www.EngineeringBooksPdf.com

www.EngineeringBooksPdf.com

CHAPTER

SEVEN

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* before going through these advanced case studies.

Case Studies	Type	Int.	Keywords
Portofolio Optimization	CQO	NO	Markowitz, Slippage, Market Impact

7.1 Portfolio Optimization

In this section the Markowitz portfolio optimization problem and variants are implemented using the **MOSEK** optimizer API.

Subsequently the following MATLAB inspired notation will be employed. The $:$ operator is used as follows

$$i : j = \{i, i+1, \ldots, j\}$$

and hence

$$x_{2:4} = \begin{bmatrix} x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

If x and y are two column vectors, then

$$[x; y] = \begin{bmatrix} x \\ y \end{bmatrix}$$

Furthermore, if $f \in \mathbb{R}^{m \times n}$ then

$$f(:) = \begin{bmatrix} f_{1,1} \\ f_{2,1} \\ \vdots \\ f_{m-1,n} \\ \vdots \\ f_{m,n} \end{bmatrix}$$

i.e. $f(:)$ stacks the columns of the matrix f.

7.1.1 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

$$\mu = \mathbf{E}r$$

www.EngineeringBooksPdf.com

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 (or risk)

$$(y - \mathbf{E}y)^2 = x^T \Sigma x.$$

The problem facing the investor is to rebalance the portfolio to achieve a good compromise between risk and expected return, e.g., maximize the expected return subject to a budget constraint and an upper bound (denoted γ) on the tolerable risk. This leads to the optimization problem

$$\begin{array}{lrcl} \text{maximize} & \mu^T x & & \\ \text{subject to} & e^T x & = & w + e^T x^0, \\ & x^T \Sigma x & \leq & \gamma^2, \\ & x & \geq & 0. \end{array} \tag{7.1}$$

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 interpretated 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, or the risk, is bounded by γ^2. Therefore, γ specifies an upper bound of the standard deviation 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

$$\Sigma = GG^T. \tag{7.2}$$

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 Section *7.1.3*.

For a given G we have that

$$\begin{array}{rcl} x^T \Sigma x & = & x^T G G^T x \\ & = & \left\| G^T x \right\|^2. \end{array}$$

www.EngineeringBooksPdf.com

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 (7.1) can be written as

$$\begin{array}{lrcl} \text{maximize} & \mu^T x & & \\ \text{subject to} & e^T x & = & w + e^T x^0, \\ & [\gamma; G^T x] & \in & \mathcal{Q}^{n+1}, \\ & x & \geq & 0, \end{array} \tag{7.3}$$

which is a conic quadratic optimization problem that can easily be solved using **MOSEK**.

Subsequently we will use the example data

$$\mu = \begin{bmatrix} 0.1073 \\ 0.0737 \\ 0.0627 \end{bmatrix}$$

and

$$\Sigma = 0.1 \begin{bmatrix} 0.2778 & 0.0387 & 0.0021 \\ 0.0387 & 0.1112 & -0.0020 \\ 0.0021 & -0.0020 & 0.0115 \end{bmatrix}$$

This implies

$$G^T = \sqrt{0.1} \begin{bmatrix} 0.5271 & 0.0734 & 0.0040 \\ 0 & 0.3253 & -0.0070 \\ 0 & 0 & 0.1069 \end{bmatrix}$$

using 5 figures of accuracy. Moreover, let

$$x^0 = \begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \end{bmatrix}$$

and

$$w = 1.0.$$

The data has been taken from *[CT07]*.

Why a Conic Formulation?

Problem (7.1) is a convex quadratically constrained optimization problems that can be solved directly using **MOSEK**, then why reformulate it as a conic quadratic optimization problem? The main reason for choosing a conic model is that it is more robust and usually leads to a shorter solution times. For instance it is not always easy to determine whether the Q matrix in (7.1) is positive semidefinite due to the presence of rounding errors. It is also very easy to make a mistake so Q becomes indefinite. These causes of problems are completely eliminated in the conic formulation.

Moreover, observe the constraint

$$\left\| G^T x \right\| \leq \gamma$$

is nicer than

$$x^T \Sigma x \leq \gamma^2$$

for small and values of γ. For instance assume a γ of 10000 then γ^2 would 1.0e8 which introduces a scaling issue in the model. Hence, using conic formulation it is possible to work with the standard deviation instead of the variance, which usually gives rise to a better scaled model.

www.EngineeringBooksPdf.com

Implementing the Portfolio Model

Model (7.3) can not be implemented as stated using the **MOSEK** optimizer API because the API requires the problem to be on the form

$$\begin{array}{lccccc} \text{maximize} & & & c^T\hat{x} & & \\ \text{subject to} & l^c & \leq & A\hat{x} & \leq & u^c, \\ & l^x & \leq & \hat{x} & \leq & u^x, \\ & & & \hat{x} \in \mathcal{K}. & & \end{array} \tag{7.4}$$

where $\hat{x}$ is referred to as the API variable.

The first step in bringing (7.3) to the form (7.4) is the reformulation

$$\begin{array}{lccc} \text{maximize} & \mu^T x & & \\ \text{subject to} & e^T x & = & w + e^T x^0, \\ & G^T x - t & = & 0 \\ & [s;t] & \in & \mathcal{Q}^{n+1}, \\ & x & \geq & 0, \\ & s & 0. & \end{array} \tag{7.5}$$

where s is an additional scalar variable and t is a n dimensional vector variable. The next step is to define a mapping of the variables

$$\hat{x} = [x;s;t] = \begin{bmatrix} x \\ s \\ t \end{bmatrix}. \tag{7.6}$$

Hence, the API variable $\hat{x}$ is concatenation of model variables x, s and t. In Table 7.1 the details of the concatenation are specified.

Table 7.1: Storage layout of the $\hat{x}$ variable.

Variable	Length	Offset
x	n	1
s	1	n+1
t	n	n+2

For instance it can be seen that

$$\hat{x}_{n+2} = t_1.$$

because the offset of the t variable is $n+2$.

Given the ordering of the variables specified by (7.6) the data should be defined as follows

$$\begin{array}{rcl} c & = & \begin{bmatrix} \mu^T & 0 & 0_{n,1} \end{bmatrix}^T, \\ A & = & \begin{bmatrix} e^T & 0 & 0_{n,1} \\ G^T & 0_{n,1} & -I_n \end{bmatrix}, \\ l^c & = & \begin{bmatrix} w + e^T x^0 & 0_{1,n} \end{bmatrix}^T, \\ u^c & = & \begin{bmatrix} w + e^T x^0 & 0_{1,n} \end{bmatrix}^T, \\ l^x & = & \begin{bmatrix} 0_{1,n} & \gamma & -\infty_{n,1} \end{bmatrix}^T, \\ u^x & = & \begin{bmatrix} \infty_{n,1} & \gamma & \infty_{n,1} \end{bmatrix}^T. \end{array}$$

www.EngineeringBooksPdf.com

The next step is to consider how the columns of A is defined. The following pseudo code

$$
\begin{array}{ll}
for & j = 1 : n \\
 & \hat{x}_j = x_j \\
 & A_{1,j} = 1.0 \\
 & A_{2:(n+1),j} = G^T_{j,1:n}
\end{array}
$$

$$\hat{x}_{n+1} = s$$

$$
\begin{array}{ll}
for & j = 1 : n \\
 & \hat{x}_{n+1+j} = t_j \\
 & A_{n+1+j,n+1+j} = -1.0
\end{array}
$$

show how to construct each column of A.

In the above discussion index origin 1 is employed, i.e., the first position in a vector is 1. The Python programming language employs 0 as index origin and that should be kept in mind when reading the example code.

```
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

  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
          for j in range(0,n):
              rtemp += x0[j]

          # 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(1+2*n)

          # Offset of variables into the API variable.
          offsetx = 0
          offsets = n
          offsett = n+1
```


www.EngineeringBooksPdf.com

```
        # 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.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))

        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,1)

        # 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 above code produce the result

Listing 7.1: Output from the solver.

```
Interior-point solution summary
Problem status  : PRIMAL_AND_DUAL_FEASIBLE
Solution status : OPTIMAL
Primal.  obj: 7.4766497707e-002   Viol.  con: 2e-008   var: 0e+000   cones: 3e-009
Dual.    obj: 7.4766522618e-002   Viol.  con: 0e+000   var: 4e-008   cones: 0e+000

Expected return 7.476650e-002 for gamma 5.000000e-002
```

The source code should be self-explanatory but a few comments are nevertheless in place.

www.EngineeringBooksPdf.com

In the lines

```
# Offset of variables into the API variable.
offsetx = 0
offsets = n
offsett = n+1
```

offsets into the **MOSEK** API variables are stored and those offsets are used later. The code

```
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.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))
```

sets up the data for `x` variables. For instance

```
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.

Debugging Tips

Implementing an optimization model in optimizer can be cumbersome and error-prone and it is very easy to make mistakes. In order to check the implemented code for mistakes it is very useful to dump the problem to a file in a human readable form for visual inspection. The line

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

does that and this will produce a file with the content

Listing 7.2: Problem (7.5) stored in OPF format.

```
[comment]
   Written by MOSEK version 7.0.0.86
   Date 01-10-13
   Time 08:43:21
[/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-001 'x[1]' + 7.37e-002 'x[2]' + 6.270000000000001e-002 'x[3]'
```


www.EngineeringBooksPdf.com

```
[/objective]

[constraints]
  [con 'budget']  'x[1]' + 'x[2]' + 'x[3]' = 1e+000 [/con]
  [con 'GT[1]']  1.667e-001 'x[1]' + 2.32e-002 'x[2]' + 1.3e-003 'x[3]' - 't[1]' = 0e+000 [/
↪con]
  [con 'GT[2]']  1.033e-001 'x[2]' - 2.2e-003 'x[3]' - 't[2]' = 0e+000 [/con]
  [con 'GT[3]']  3.38e-002 'x[3]' - 't[3]' = 0e+000 [/con]
[/constraints]

[bounds]
  [b]          0 <= * [/b]
  [b]               s =  5e-002 [/b]
  [b]               't[1]','t[2]','t[3]' free [/b]
  [cone quad 'stddev'] s, 't[1]', 't[2]', 't[3]' [/cone]
[/bounds]
```

Observe that since the API variables have been given meaningful names it is easy to see the model is correct.

7.1.2 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 then the problem

$$\begin{array}{lccc} \text{maximize} & \mu^T x - \alpha s & & \\ \text{subject to} & e^T x & = & w + e^T x^0, \\ & [s; G^T x] & \in & \mathcal{Q}^{n+1}, \\ & x & \geq & 0. \end{array} \tag{7.7}$$

computes efficient portfolios. Note that the objective maximizes the expected return while maximizing $-\alpha$ times the standard deviation. Hence, the standard deviation is minimized while α specifies the tradeoff between expected return and risk.

Ideally the problem (7.7) should be solved for all values $\alpha \geq 0$ but in practice that is computationally too costly.

Using the example data from Section *7.1.1*, the optimal values of return and risk for several αs are listed below:

Listing 7.3: Results obtained solving problem (7.7) for different values of α.

```
   alpha          exp ret        std dev
   0.000e+000     1.073e-001     7.261e-001
   2.500e-001     1.033e-001     1.499e-001
   5.000e-001     6.976e-002     3.735e-002
   7.500e-001     6.766e-002     3.383e-002
   1.000e+000     6.679e-002     3.281e-002
   1.500e+000     6.599e-002     3.214e-002
   2.000e+000     6.560e-002     3.192e-002
   2.500e+000     6.537e-002     3.181e-002
   3.000e+000     6.522e-002     3.176e-002
   3.500e+000     6.512e-002     3.173e-002
   4.000e+000     6.503e-002     3.170e-002
   4.500e+000     6.497e-002     3.169e-002
```


www.EngineeringBooksPdf.com

The example code in Listing 7.4 demonstrates how to compute the efficient portfolios for several values of α.

Listing 7.4: Code implementing model (7.7)

```
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
  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
          for j in range(0,n):
              rtemp += x0[j]

          # 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(1+2*n)

          offsetx = 0   # Offset of variable x into the API variable.
          offsets = n   # Offset of variable x into the API variable.
          offsett = n+1 # Offset of variable t 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.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.fr,gamma,gamma)
          task.putvarname(offsets+0,"s")
```


www.EngineeringBooksPdf.com

```
        # 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))

        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)

        for alpha in alphas:
            # Dump the problem to a human readable OPF file.
            #task.writedata("dump.opf")

            task.putcj(offsets+0,-alpha);

            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, mosek.solsta.near_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("\nExpected return %e for gamma %e" % (expret,stddev[0])),
            else:
                print("An error occurred when solving for alpha=%e\n" % alpha)
```

7.1.3 Improving the Computational Efficiency

In practice it is often important to solve the portfolio problem in a short amount of time; this section it is discusses what can be done at the modelling stage to improve the computational efficiency.

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 number nonzeros used to represent the problem. Indeed it is often better to focus at the number of nonzeros in G (see (7.2)) 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 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 factor model and usually p tends be a small number, say less than 100, independent of n.

www.EngineeringBooksPdf.com

One possible choice for G is the Cholesky factorization of Σ which requires storage proportional to $n(n+1)/2$. However, another choice is

$$G^T = \begin{bmatrix} D^{1/2} \\ V^T \end{bmatrix}$$

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 then the difference in storage requirements is 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 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.

7.1.4 Slippage Cost

The basic Markowitz portfolio model assumes that there are no costs associated with trading the assets and that the returns of the assets is independent of the amount traded. None of those assumptions are usually valid in practice. Therefore, a more realistic model is

$$\begin{array}{lccc} \text{maximize} & \mu^T x & & \\ \text{subject to} & e^T x + \sum_{j=1}^n C_j(x_j - x_j^0) & = & w + e^T x^0, \\ & x^T \Sigma x & \leq & \gamma^2, \\ & x & \geq & 0, \end{array} \tag{7.8}$$

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.

Market Impact Costs

If the initial wealth is fairly small and short selling is not allowed, then the holdings will be small. Therefore, the amount traded of each asset must also be small. Hence, it is reasonable to assume that the prices of the assets is independent of the amount traded. However, if a large volume of an assert is sold or purchased it can be expected that the price change and hence the expected return also change. This effect is called market impact costs. It is common to assume that market impact costs for asset j can be modelled by

$$m_j \sqrt{|x_j - x_j^0|}$$

where m_j is a constant that is estimated in some way. See *[GK00]*[p. 452] for details. To summarize then

$$C_j(x_j - x_j^0) = m_j |x_j - x_j^0| \sqrt{|x_j - x_j^0|} = m_j |x_j - x_j^0|^{3/2}.$$

From *[MOSEKApS12]* it is known

$$\{(c,z) : c \geq z^{3/2}, z \geq 0\} = \{(c,z) : [v;c;z], [z;1/8;v] \in \mathcal{Q}_r^3\}$$

www.EngineeringBooksPdf.com

where $\mathcal{Q}_r^3$ is the 3 dimensional rotated quadratic cone implying

$$\begin{array}{lcl} z_j & = & |x_j - x_j^0|, \\ [v_j; c_j; z_j], [z_j; 1/8; v_j] & \in & \mathcal{Q}_r^3, \\ \sum_{j=1}^n C_j(x_j - x_j^0) & = & \sum_{j=1}^n c_j. \end{array}$$

Unfortunately this set of constraints is nonconvex due to the constraint

$$z_j = |x_j - x_j^0| \tag{7.9}$$

but in many cases that constraint can safely be replaced by the relaxed constraint

$$z_j \geq |x_j - x_j^0| \tag{7.10}$$

which is convex. If for instance the universe of assets contains a risk free asset with a positive return then

$$z_j > |x_j - x_j^0| \tag{7.11}$$

cannot hold for an optimal solution because that would imply the solution is not optimal.

Now assume that the optimal solution has the property that (7.11) holds 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 then the only way to obtain a small risk is to get rid of some of the assets by generating transaction costs. Here it is assumed this is not the case and hence the models (7.9) and (7.10) are equivalent.

Formula (7.10) is replaced by constraints

$$\begin{array}{rcl} z_j & \geq & x_j - x_j^0, \\ z_j & \geq & -(x_j - x_j^0). \end{array} \tag{7.12}$$

Now we have

$$\begin{array}{lrcll} \text{maximize} & \mu^T x & & & \\ \text{subject to} & e^T x + m^T c & = & w + e^T x^0, & \\ & z_j & \geq & x_j - x_j^0, & j = 1, \ldots, n, \\ & z_j & \geq & x_j^0 - x_j, & j = 1, \ldots, n, \\ & [\gamma; G^T x] & \in & \mathcal{Q}^{n+1}, & \\ & [v_j; c_j; z_j] & \in & \mathcal{Q}_r^3, & j = 1, \ldots, n, \\ & [z_j; 1/8; v_j] & \in & \mathcal{Q}_r^3, & j = 1, \ldots, n, \\ & x & \geq & 0. & \end{array} \tag{7.13}$$

The revised budget constraint

$$e^T x = w + e^T x^0 - m^T c$$

specifies that the total investment must be equal to the initial wealth minus the transaction costs. Moreover, observe the variables v and z are some auxiliary variables that model the market impact cost. Indeed it holds

$$z_j \geq |x_j - x_j^0|$$

and

$$c_j \geq z_j^{3/2}.$$

Before proceeding it should be mentioned that transaction costs of the form

$$c_j \geq z_j^{p/q}$$

www.EngineeringBooksPdf.com

where p and q are both integers and $p \geq q$ can be modelled using quadratic cones. See *[MOSEKApS12]* for details.

One more reformulation of (7.13) is needed,

$$\begin{array}{lccll} \text{maximize} & \mu^T x & & & \\ \text{subject to} & e^T x + m^T c & = & w + e^T x^0, & \\ & G^T x - t & = & 0, & \\ & z_j - x_j & \geq & -x_j^0, & j = 1, \ldots, n, \\ & z_j + x_j & \geq & x_j^0, & j = 1, \ldots, n, \\ & [v_j; c_j; z_j] - f_{j,1:3} & = & 0, & j = 1, \ldots, n, \\ & [z_j; 0; v_j] - g_{j,1:3} & = & [0; -1/8; 0], & j = 1, \ldots, n, \\ & [s; t] & \in & \mathcal{Q}^{n+1}, & \\ & f_{j,1:3}^T & \in & \mathcal{Q}_r^3, & j = 1, \ldots, n, \\ & g_{j,1:3}^T & \in & \mathcal{Q}_r^3, & j = 1, \ldots, n, \\ & x & \geq & 0, & \\ & s & = & \gamma, & \end{array} \tag{7.14}$$

where $f, g \in \mathbb{R}^{n \times 3}$. These additional variables f and g are only introduced to bring the problem on the API standard form.

The formulation (7.14)) is not the most compact possible. However, the **MOSEK** presolve will automatically make it more compact and since it is easier to implement (7.14) than a more compact form then the form (7.14) is preferred.

The first step in developing the optimizer API implementation is to chose an ordering of the variables. In this case the ordering

$$\hat{x} = \begin{bmatrix} x \\ s \\ t \\ c \\ v \\ z \\ f^T(:) \\ g^T(:) \end{bmatrix}$$

will be used. Note $f^T(:)$ means the rows of f are transposed and stacked on top of each other to form a long column vector. Table 7.2 shows the mapping between the $\hat{x}$ and the model variables.

Table 7.2: Storage layout for the $\hat{x}$

Variable	Length	Offset
x	n	1
s	1	n+1
t	n	n+2
c	n	2n+2
v	n	3n+2
z	n	4n+2
$f(:)^T$	3n	7n+2
$g(:)^T$	3n	10n+2

The next step is to consider how the columns of A is defined. Reusing the idea in Section *7.1.1* then the

www.EngineeringBooksPdf.com

following pseudo code describes the setup of A.

$$
\begin{array}{ll}
for & j = 1 : n \\
 & \hat{x}_j = x_j \\
 & A_{1,j} = 1.0 \\
 & A_{2:n+1,j} = G^T_{j,1:n} \\
 & A_{n+1+j,j} = -1.0 \\
 & A_{2n+1+j,j} = 1.0
\end{array}
$$

$$\hat{x}_{n+1} = s$$

$$
\begin{array}{ll}
for & j = 1 : n \\
 & \hat{x}_{n+1+j} = t_j \\
 & A_{1+j,n+1+j} = -1.0
\end{array}
$$

$$
\begin{array}{ll}
for & j = 1 : n \\
 & \hat{x}_{2n+1+j} = c_j \\
 & A_{1,2n+1+j} = m_j \\
 & A_{3n+1+3(j-1)+2,2n+1+j} = 1.0
\end{array}
$$

$$
\begin{array}{ll}
for & j = 1 : n \\
 & \hat{x}_{3n+1+j} = v_j \\
 & A_{3n+1+3(j-1)+1,3n+1+j} = 1.0 \\
 & A_{6n+1+3(j-1)+3,3n+1+j} = 1.0
\end{array}
$$

$$
\begin{array}{ll}
for & j = 1 : n \\
 & \hat{x}_{4n+1+j} = z_j \\
 & A_{1+n+j,4n+1+j} = 1.0 \\
 & A_{1+2n+j,4n+1+j} = 1.0 \\
 & A_{3n+1+3(j-1)+3,4n+1+j} = 1.0 \\
 & A_{6n+1+3(j-1)+1,4n+1+j} = 1.0
\end{array}
$$

$$
\begin{array}{ll}
for & j = 1 : n \\
 & \hat{x}_{7n+1+3(j-1)+1} = f_{j,1} \\
 & A_{3n+1+3(j-1)+1,7n+(3(j-1)+1} = -1.0 \\
 & \hat{x}_{7n+1+3(j-1)+2} = f_{j,2} \\
 & A_{3n+1+3(j-1)+2,7n+(3(j-1)+2} = -1.0 \\
 & \hat{x}_{7n+1+3(j-1)+3} = f_{j,3} \\
 & A_{3n+1+3(j-1)+3,7n+(3(j-1)+3} = -1.0
\end{array}
$$

$$
\begin{array}{ll}
for & j = 1 : n \\
 & \hat{x}_{10n+1+3(j-1)+1} = g_{j,1} \\
 & A_{6n+1+3(j-1)+1,7n+(3(j-1)+1} = -1.0 \\
 & \hat{x}_{10n+1+3(j-1)+2} = g_{j,2} \\
 & A_{6n+1+3(j-1)+2,7n+(3(j-1)+2} = -1.0 \\
 & \hat{x}_{10n+1+3(j-1)+3} = g_{j,3} \\
 & A_{6n+1+3(j-1)+3,7n+(3(j-1)+3} = -1.0
\end{array}
$$

The example code in Listing 7.5 demonstrates how to implement the model (7.14).

Listing 7.5: Code implementing model (7.14).

```
import mosek


def streamprinter(text):
    print("%s" % text),

if __name__ == '__main__':
```


www.EngineeringBooksPdf.com

```
  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+9*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))

          task.putconboundlist(range(1+3*n,1+3*n+3*n),3*n*[mosek.boundkey.fx],3*n*[0.],3*n*[0.
↪0])
          for i in range(0,n):
              for k in range(0,n):
                  task.putconname(1+3*n+3*i+k,"f[%d,%d]" % (1+i,1+k))

          task.putconboundlist(range(1+6*n,1+9*n),3*n*[mosek.boundkey.fx],
                               3*[0.0, -1.0/8.0, 0.0],3*[0.0, -1.0/8.0, 0.0])
          for i in range(0,n) :
              for k in range(0,n):
                  task.putconname(1+6*n+3*i+k,"g[%d,%d]" % (1+i,1+k))

          # Offset of variables into the API variable.
          offsetx = 0
          offsets = n
          offsett = n+1
          offsetc = 2*n+1
          offsetv = 3*n+1
          offsetz = 4*n+1
          offsetf = 5*n+1
          offsetg = 8*n+1

          # Variables.
          task.appendvars(1+11*n)
```


www.EngineeringBooksPdf.com

```
        # 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.putaijlist(range(1+3*n+1,1+6*n+1,3),range(offsetc,offsetc+n),n*[1.0])
        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))

        # v variables.
        task.putaijlist(range(1+3*n+0,1+6*n+0,3),range(offsetv,offsetv+n),n*[1.0])
        task.putaijlist(range(1+6*n+2,1+9*n+2,3),range(offsetv,offsetv+n),n*[1.0])
        task.putvarboundlist(range(offsetv,offsetv+n),n*[mosek.boundkey.fr],n*[-inf],n*[inf])
        for j in range(0,n):
            task.putvarname(offsetv+j,"v[%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.putaijlist(range(1+3*n+2,1+6*n+2,3),range(offsetz,offsetz+n),n*[1.0])
        task.putaijlist(range(1+6*n+0,1+9*n+0,3),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.
        for j in range(0,n):
            for k in range(0,n):
                task.putaij(1+3*n+3*j+k,offsetf+3*j+k,-1.0)
                task.putvarbound(offsetf+3*j+k,mosek.boundkey.fr,-inf,inf)
                task.putvarname(offsetf+3*j+k,"f[%d,%d]" % (1+j,1+k))

        # g variables.
        for j in range(0,n):
            for k in range(0,n):
                task.putaij(1+6*n+3*j+k,offsetg+3*j+k,-1.0)
                task.putvarbound(offsetg+3*j+k,mosek.boundkey.fr,-inf,inf)
                task.putvarname(offsetg+3*j+k,"g[%d,%d]" % (1+j,1+k))

        task.appendcone(mosek.conetype.quad,0.0,[offsets] + list(range(offsett,offsett+n)))
```


www.EngineeringBooksPdf.com

```
        task.putconename(0,"stddev")

        for k in range(0,n):
            task.appendconeseq(mosek.conetype.rquad,0.0,3,offsetf+3*k)
            task.putconename(1+k,"f[%d]" % (1+k))

        for k in range(0,n):
            task.appendconeseq(mosek.conetype.rquad,0.0,3,offsetg+3*k)
            task.putconename(1+n+k,"g[%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: 7.4390654948e-002   Viol.  con: 2e-007   var: 0e+000   cones: 2e-009
Dual.    obj: 7.4390665143e-002   Viol.  con: 2e-008   var: 2e-008   cones: 0e+000
Expected return 7,439065E-002 for gamma 5,000000E-002
```

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

Listing 7.6: OPF file for problem (7.14).

```
[comment]
   Written by MOSEK version 7.0.0.86
   Date 01-10-13
   Time 08:59:30
[/comment]

[hints]
  [hint NUMVAR] 34 [/hint]
  [hint NUMCON] 28 [/hint]
  [hint NUMANZ] 60 [/hint]
  [hint NUMQNZ] 0 [/hint]
  [hint NUMCONE] 7 [/hint]
[/hints]

[variables disallow_new_variables]
```


www.EngineeringBooksPdf.com

```
  'x[1]' 'x[2]' 'x[3]' s 't[1]'
  't[2]' 't[3]' 'c[1]' 'c[2]' 'c[3]'
  'v[1]' 'v[2]' 'v[3]' 'z[1]' 'z[2]'
  'z[3]' 'f[1,1]' 'f[1,2]' 'f[1,3]' 'f[2,1]'
  'f[2,2]' 'f[2,3]' 'f[3,1]' 'f[3,2]' 'f[3,3]'
  'g[1,1]' 'g[1,2]' 'g[1,3]' 'g[2,1]' 'g[2,2]'
  'g[2,3]' 'g[3,1]' 'g[3,2]' 'g[3,3]'
[/variables]

[objective maximize]
   1.073e-001 'x[1]' + 7.37e-002 'x[2]' + 6.270000000000001e-002 'x[3]'
[/objective]

[constraints]
  [con 'budget']  'x[1]' + 'x[2]' + 'x[3]' + 1e-002 'c[1]' + 1e-002 'c[2]'
     + 1e-002 'c[3]' = 1e+000 [/con]
  [con 'GT[1]']  1.667e-001 'x[1]' + 2.32e-002 'x[2]' + 1.3e-003 'x[3]' - 't[1]' = 0e+000 [/
↪con]
  [con 'GT[2]']  1.033e-001 'x[2]' - 2.2e-003 'x[3]' - 't[2]' = 0e+000 [/con]
  [con 'GT[3]']  3.38e-002 'x[3]' - 't[3]' = 0e+000 [/con]
  [con 'zabs1[1]'] 0e+000 <= - 'x[1]' + 'z[1]' [/con]
  [con 'zabs1[2]'] 0e+000 <= - 'x[2]' + 'z[2]' [/con]
  [con 'zabs1[3]'] 0e+000 <= - 'x[3]' + 'z[3]' [/con]
  [con 'zabs2[1]'] 0e+000 <= 'x[1]' + 'z[1]' [/con]
  [con 'zabs2[2]'] 0e+000 <= 'x[2]' + 'z[2]' [/con]
  [con 'zabs2[3]'] 0e+000 <= 'x[3]' + 'z[3]' [/con]
  [con 'f[1,1]']  'v[1]' - 'f[1,1]' = 0e+000 [/con]
  [con 'f[1,2]']  'c[1]' - 'f[1,2]' = 0e+000 [/con]
  [con 'f[1,3]']  'z[1]' - 'f[1,3]' = 0e+000 [/con]
  [con 'f[2,1]']  'v[2]' - 'f[2,1]' = 0e+000 [/con]
  [con 'f[2,2]']  'c[2]' - 'f[2,2]' = 0e+000 [/con]
  [con 'f[2,3]']  'z[2]' - 'f[2,3]' = 0e+000 [/con]
  [con 'f[3,1]']  'v[3]' - 'f[3,1]' = 0e+000 [/con]
  [con 'f[3,2]']  'c[3]' - 'f[3,2]' = 0e+000 [/con]
  [con 'f[3,3]']  'z[3]' - 'f[3,3]' = 0e+000 [/con]
  [con 'g[1,1]']  'z[1]' - 'g[1,1]' = 0e+000 [/con]
  [con 'g[1,2]']  - 'g[1,2]' = -1.25e-001 [/con]
  [con 'g[1,3]']  'v[1]' - 'g[1,3]' = 0e+000 [/con]
  [con 'g[2,1]']  'z[2]' - 'g[2,1]' = 0e+000 [/con]
  [con 'g[2,2]']  - 'g[2,2]' = -1.25e-001 [/con]
  [con 'g[2,3]']  'v[2]' - 'g[2,3]' = 0e+000 [/con]
  [con 'g[3,1]']  'z[3]' - 'g[3,1]' = 0e+000 [/con]
  [con 'g[3,2]']  - 'g[3,2]' = -1.25e-001 [/con]
  [con 'g[3,3]']  'v[3]' - 'g[3,3]' = 0e+000 [/con]
[/constraints]

[bounds]
  [b]          0 <= * [/b]
  [b]               s =  5e-002 [/b]
  [b]               't[1]','t[2]','t[3]','c[1]','c[2]','c[3]' free [/b]
  [b]               'v[1]','v[2]','v[3]','z[1]','z[2]','z[3]' free [/b]
  [b]               'f[1,1]','f[1,2]','f[1,3]','f[2,1]','f[2,2]','f[2,3]' free [/b]
  [b]               'f[3,1]','f[3,2]','f[3,3]','g[1,1]','g[1,2]','g[1,3]' free [/b]
  [b]               'g[2,1]','g[2,2]','g[2,3]','g[3,1]','g[3,2]','g[3,3]' free [/b]
  [cone quad 'stddev'] s, 't[1]', 't[2]', 't[3]' [/cone]
  [cone rquad 'f[1]'] 'f[1,1]', 'f[1,2]', 'f[1,3]' [/cone]
  [cone rquad 'f[2]'] 'f[2,1]', 'f[2,2]', 'f[2,3]' [/cone]
  [cone rquad 'f[3]'] 'f[3,1]', 'f[3,2]', 'f[3,3]' [/cone]
  [cone rquad 'g[1]'] 'g[1,1]', 'g[1,2]', 'g[1,3]' [/cone]
  [cone rquad 'g[2]'] 'g[2,1]', 'g[2,2]', 'g[2,3]' [/cone]
  [cone rquad 'g[3]'] 'g[3,1]', 'g[3,2]', 'g[3,3]' [/cone]
[/bounds]
```


www.EngineeringBooksPdf.com

The file verifies that the correct problem has been setup.

www.EngineeringBooksPdf.com

www.EngineeringBooksPdf.com

CHAPTER

EIGHT

ERRORS AND WARNINGS

Interaction between **MOSEK** and the user is not always succesfull and critical situation may arise for several reasons: wrong input data, unexpected numerical issues, not enough memory, etc. **MOSEK** reports these events to the user making the following distinction:

- *Warning*: it informs the user about a non critical but important event that will not prevent the solver execution. When a warning arises the final results may not be the expected.
- *Error*: it informs the user about a critical and possibly unrecoverable event. The required operation will not be performed correctly.

Therefore errors and warnings must be handled carefully to use the solver in a safe way.

8.1 Warnings

Warning messages generated by **MOSEK** should in general never be ignored. Despite not being critical, they provide useful information and often are the key to understand how to improve the solver performance or solve numerical issues. For this reason, it is a good practice to start working with a verbose output on the screen (see Section *9*) in order to spot possible warnings.

Typically warnings involve

- Numerical criticalities in the optimization model: for instance if very large upper bound on a constraint is specified, the solver will notify the user with a message like the following

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

- Wrong parameter values

Ideally the solver should not rise any warning. The user should work on the optimization model and the way the solver is called in order to remove all warnings. We recommend to ignore warnings only if

- they are well understood by the user and therefore deliberately ignored;
- they are related to non critical operations (as for instance variable naming) and the results obtained are correct.

In these cases, warnings can be suppressed by setting the *iparam.max_num_warnings* parameter to zero.

8.2 Errors

Errors are the result of

- wrong input data (too large variable index for instance)
- system limitation (for instance not enough memory available)
- bugs in the API (see Section *1.2*)

www.EngineeringBooksPdf.com

Most functions in the **MOSEK** Optimizer API for Python return a *response code* which indicates whether an error occurred. It is recommended to check to the response code and in case it is indicating an error then an appropriate action should be taken.

8.2.1 Checking for Memory Leaks and Overwrites

If you suspect that **MOSEK** or your own application incorrectly overwrites memory or leaks memory, we suggest you use external tools such as Intel Inspector for C and .NET, or Valgrind to pinpoint the cause of the problem.

8.2.2 Debugging Tips

Turn on logging

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

See example in Section *3.2* for instructions on turning log output on. You should also always cache and handle any exceptions thrown by **MOSEK**.

Please refer to Section *9* for further information.

Dump problem to OPF file

If something is wrong with a problem or a solution, one option is to output the problem to an *OPF file* and inspect it by hand. Use the *task.writedata* function to write a task to a file immediately before optimizing, for example as follows:

```
task.writedata('taskdump.opf')
task.optimizetrm()
```

This will write the problem in `task` to the file `taskdump.opf`. Inspecting the text file `taskdump.opf` may reveal what is wrong in the problem setup.

www.EngineeringBooksPdf.com

CHAPTER

NINE

MANAGING I/O

The main purpose of this chapter is to give an overview on the logging and I/O features provided by the **MOSEK** package.

- Section *9.1* contains information about the log streams provided by **MOSEK**.
- File I/O is discussed in Section *9.2*.
- How to tune the logging verbosity is the topic of Section *9.3*.

9.1 Stream I/O

MOSEK execution produces a certain amount of loggging at environment and task level. This means that the logging from each environement and task can be isolated from the others.

The log messages are partitioned in three streams:

- **messages**
- **warnings**
- **errors**

These streams are aggregated in the **log** stream. See *streamtype*.

Each stream can be redirected either to a user defined function or to a file.

Log stream to function

Link a custom function to a stream is particularly useful to generate specialize output.

Text written to specific **MOSEK** streams can be intercepted by attaching a stream handler to the specific stream. A stream handler is simply a function that accepts one string argument, for instance

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

To attach a stream callback to a *Task*, use:

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

In this example we attached to the `log` stream; see *streamtype* for other options.

To attach a stream callback to a *Task*, use:

The stream can be detached by calling

```
task.set_Stream(None)
```


www.EngineeringBooksPdf.com

Note: As for `Progress` callbacks, it is necessary to hold a reference to the callback function for the duration of the lifetime of the *Task* or *Env* object.

Log stream to file

A stream can be redirected to a file passing to the solver the file name. The solver creates the file anew or append the log to an existing one. The file is closed when the task/environment is destroyed.

9.2 File I/O

MOSEK supports a range of problem and solution formats listed in Section *17*. One such format is **MOSEK**'s native binary *Task format* which supports all features that **MOSEK** supports.

The file format used in I/O operations is deduced from extension - as in `problemname.task` - unless the parameter *iparam.write_data_format* is specified to something else. Problem files with an additional `.gz` extension - as in `problemname.task.gz` - are moreover assumed to use GZIP compression, and are automatically compressed, respectively decompressed, when written or read.

Example

If something is wrong with a problem or a solution, one option is to output the problem to the human-readable *OPF format* and inspect it by hand. For instance, one may use the *task.writedata* function to write the problem to a file immediately before optimizing it:

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

This will write the problem in `task` to the file `data.opf`.

9.3 Verbosity

The logging verbosity can be controlled by setting the relevant paramenters, 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 control 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, and it is consider the default setting. When output is no more of interest, user can easily disable using *iparam.log*.

www.EngineeringBooksPdf.com

Moreover, it must be understood that larger values of *iparam.log* do not necessarily result in an increased output.

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

www.EngineeringBooksPdf.com

www.EngineeringBooksPdf.com

CHAPTER

TEN

PROBLEM FORMULATION AND SOLUTIONS

In this chapter we will discuss the following issues:

- The formal definitions of the problem types that **MOSEK** can solve.
- The solution information produced by **MOSEK**.
- The information produced by **MOSEK** if the problem is infeasible.

10.1 Linear Optimization

A linear optimization problem can be written as

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

where

- m is the number of constraints.
- n is the number of decision variables.
- $x \in \mathbb{R}^n$ is a vector of decision variables.
- $c \in \mathbb{R}^n$ is the linear part of the objective function.
- $A \in \mathbb{R}^{m \times n}$ is the constraint matrix.
- $l^c \in \mathbb{R}^m$ is the lower limit on the activity for the constraints.
- $u^c \in \mathbb{R}^m$ is the upper limit on the activity for the constraints.
- $l^x \in \mathbb{R}^n$ is the lower limit on the activity for the variables.
- $u^x \in \mathbb{R}^n$ is the upper limit on the activity for the variables.

A primal solution (x) is *(primal) feasible* if it satisfies all constraints in (10.1). If (10.1) has at least one primal feasible solution, then (10.1) is said to be (primal) feasible.

In case (10.1) does not have a feasible solution, the problem is said to be *(primal) infeasible*

10.1.1 Duality for Linear Optimization

Corresponding to the primal problem (10.1), there is a dual problem

$$\begin{array}{lcccc} \text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x + c^f & & & \\ & A^T y + s_l^x - s_u^x & = & c, & \\ \text{subject to} & -y + s_l^c - s_u^c & = & 0, & \\ & s_l^c, s_u^c, s_l^x, s_u^x & \geq & 0. & \end{array} \tag{10.2}$$

www.EngineeringBooksPdf.com

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

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

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

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

to the dual problem is feasible if it satisfies all the constraints in (10.2). If (10.2) has at least one feasible solution, then (10.2) is *(dual) feasible*, otherwise the problem is *(dual) infeasible*.

A Primal-dual Feasible Solution

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 (10.1) and $(y, s_l^c, s_u^c, s_l^x, s_u^x)$ is a solution to the corresponding dual problem (10.2).

The Duality Gap

Let

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

be a primal-dual feasible solution, and let

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

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

$$\begin{array}{l} c^T x^* + c^f - \left\{(l^c)^T (s_l^c)^* - (u^c)^T (s_u^c)^* + (l^x)^T (s_l^x)^* - (u^x)^T (s_u^x)^* + c^f\right\} \\ \quad = \sum_{i=0}^{m-1} \left[(s_l^c)_i^* ((x_i^c)^* - l_i^c) + (s_u^c)_i^* (u_i^c - (x_i^c)^*)\right] \\ \quad + \sum_{j=0}^{n-1} \left[(s_l^x)_j^* (x_j - l_j^x) + (s_u^x)_j^* (u_j^x - x_j^*)\right] \geq 0 \end{array} \tag{10.3}$$

where the first relation can be obtained by transposing and multiplying the dual constraints (10.2) 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.

An Optimal Solution

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

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

are satisfied.

If (10.1) 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.

www.EngineeringBooksPdf.com

10.1.2 Infeasibility for Linear Optimization

Primal Infeasible Problems

If the problem (10.1) is infeasible (has no feasible solution), **MOSEK** will report a certificate of primal infeasibility: The dual solution reported is the certificate of infeasibility, and the primal solution is undefined.

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

$$\begin{array}{ll} \text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x \\ \text{subject to} & \\ & \begin{array}{c} A^T y + s_l^x - s_u^x = 0, \\ -y + s_l^c - s_u^c = 0, \\ s_l^c, s_u^c, s_l^x, s_u^x \geq 0, \end{array} \end{array} \tag{10.4}$$

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 (10.4) 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 (10.4) is unbounded, and that its dual is infeasible. As the constraints to the dual of (10.4) are identical to the constraints of problem (10.1), we thus have that problem (10.1) is also infeasible.

Dual Infeasible Problems

If the problem (10.2) 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

$$\begin{array}{lccccc} \text{minimize} & & & c^T x & & \\ \text{subject to} & \hat{l}^c & \leq & Ax & \leq & \hat{u}^c, \\ & \hat{l}^x & \leq & x & \leq & \hat{u}^x, \end{array} \tag{10.5}$$

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 (10.5) is unbounded, and that its dual is infeasible. As the constraints to the dual of (10.5) are identical to the constraints of problem (10.2), we thus have that problem (10.2) is also infeasible.

Primal and Dual Infeasible Case

In case that both the primal problem (10.1) and the dual problem (10.2) are infeasible, **MOSEK** will report only one of the two possible certificates — which one is not defined (**MOSEK** returns the first certificate found).

www.EngineeringBooksPdf.com

Minimalization vs. Maximalization

When the objective sense of problem (10.1) 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 (10.2). 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 (10.3) as the primal minus the dual objective value, becomes nonpositive. It follows that the dual objective will always be greater than or equal to the primal objective. The primal infeasibility certificate will be reported by **MOSEK** as a solution to the system

$$\begin{array}{l} A^T y + s_l^x - s_u^x = 0, \\ -y + s_l^c - s_u^c = 0, \\ s_l^c, s_u^c, s_l^x, s_u^x \leq 0, \end{array} \tag{10.6}$$

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 (10.5) such that $c^T x > 0$.

10.2 Conic Quadratic Optimization

Conic quadratic optimization is an extension of linear optimization (see Section *10.1*) allowing conic domains to be specified for subsets of the problem variables. A conic quadratic optimization problem can be written as

$$\begin{array}{lccccc} \text{minimize} & & & c^T x + c^f & & \\ \text{subject to} & l^c & \leq & Ax & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x, \\ & & & x \in \mathcal{K}, & & \end{array} \tag{10.7}$$

where 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.

MOSEK supports only a limited number of cones, specifically:

- The $\mathbb{R}^n$ set.
- The quadratic cone:

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

- The rotated quadratic cone:

www.EngineeringBooksPdf.com

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

Although these cones may seem to provide only limited expressive power they can be used to model a wide range of problems as demonstrated in *[MOSEKApS12]*.

10.2.1 Duality for Conic Quadratic Optimization

The dual problem corresponding to the conic quadratic optimization problem (10.7) is given by

$$\begin{array}{ll} \text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x + c^f \\ \text{subject to} & \\ & \begin{array}{c} A^T y + s_l^x - s_u^x + s_n^x = c \\ -y + s_l^c - s_u^c = 0, \\ s_l^c, s_u^c, s_l^x, s_u^x \geq 0, \\ s_n^x \in \mathcal{K}^*, \end{array} \end{array} \tag{10.8}$$

where the dual cone $\mathcal{K}^*$ is a Cartesian product of the cones

$$\mathcal{K}^* = \mathcal{K}_1^* \times \cdots \times \mathcal{K}_p^*,$$

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

- The $\mathbb{R}^n$ set:

$$\mathcal{K}_t = \mathbb{R}^{n_t} \quad \Leftrightarrow \quad \mathcal{K}_t^* = \{ s \in \mathbb{R}^{n_t} : \quad s = 0 \}.$$

- The quadratic cone:

$$\mathcal{K}_t = \mathcal{Q}^{n_t} \quad \Leftrightarrow \quad \mathcal{K}_t^* = \mathcal{Q}^{n_t} = \left\{ s \in \mathbb{R}^{n_t} : s_1 \geq \sqrt{\sum_{j=2}^{n_t} s_j^2} \right\}.$$

- The rotated quadratic cone:

$$\mathcal{K}_t = \mathcal{Q}_r^{n_t} \quad \Leftrightarrow \quad \mathcal{K}_t^* = \mathcal{Q}_r^{n_t} = \left\{ s \in \mathbb{R}^{n_t} : 2s_1s_2 \geq \sum_{j=3}^{n_t} s_j^2, \quad s_1 \geq 0, \quad s_2 \geq 0 \right\}.$$

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

10.2.2 Infeasibility for Conic Quadratic Optimization

In case **MOSEK** finds a problem to be infeasible it reports a certificate of infeasibility. This works exactly as for linear problems (see Section *10.1.2*).

Primal Infeasible Problems

If the problem (10.7) is infeasible, **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 problem

$$\begin{array}{ll} \text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x \\ \text{subject to} & \\ & \begin{array}{lcl} A^T y + s_l^x - s_u^x + s_n^x & = & 0, \\ -y + s_l^c - s_u^c & = & 0, \\ s_l^c, s_u^c, s_l^x, s_u^x & \geq & 0, \\ s_n^x \in \mathcal{K}^*, & & \end{array} \end{array}$$

such that the objective value is strictly positive.

www.EngineeringBooksPdf.com

Dual infeasible problems

If the problem (10.8) is infeasible, **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 problem

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

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 the objective value is strictly negative.

10.3 Semidefinite Optimization

Semidefinite optimization is an extension of conic quadratic optimization (see Section *10.2*) allowing positive semidefinite matrix variables to be used in addition to the usual scalar variables. A semidefinite optimization problem can be written as

$$\begin{array}{lccccll} \text{minimize} & & & \sum_{j=0}^{n-1} c_j x_j + \sum_{j=0}^{p-1} \langle \overline{C}_j, \overline{X}_j \rangle + c^f & & & \\ \text{subject to} & l_i^c & \leq & \sum_{j=0}^{n-1} a_{ij} x_j + \sum_{j=0}^{p-1} \langle \overline{A}_{ij}, \overline{X}_j \rangle & \leq & u_i^c, & i = 0, \ldots, m-1 \\ & l_j^x & \leq & x_j & \leq & u_j^x, & j = 0, \ldots, n-1 \\ & & & x \in \mathcal{K}, \overline{X}_j \in \mathcal{S}_+^{r_j}, & & & j = 0, \ldots, p-1 \end{array} \tag{10.9}$$

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

$$\langle U, V \rangle := \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} U_{ij} V_{ij}.$$

With semidefinite optimization we can model a wide range of problems as demonstrated in *[MOSEKApS12]*.

10.3.1 Duality for Semidefinite Optimization

The dual problem corresponding to the semidefinite optimization problem (10.9) is given by

$$\begin{array}{ll} \text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x + c^f \\ \text{subject to} & \\ & c - A^T y + s_u^x - s_l^x = s_n^x, \\ & \overline{C}_j - \sum_{i=0}^{m} y_i \overline{A}_{ij} = \overline{S}_j, \qquad j = 0, \ldots, p-1 \\ & s_l^c - s_u^c = y, \\ & s_l^c, s_u^c, s_l^x, s_u^x \geq 0, \\ & s_n^x \in \mathcal{K}^*, \quad \overline{S}_j \in \mathcal{S}_+^{r_j}, \qquad j = 0, \ldots, p-1 \end{array} \tag{10.10}$$

www.EngineeringBooksPdf.com

where $A \in \mathbb{R}^{m\times n}$, $A_{ij} = a_{ij}$, which is similar to the dual problem for conic quadratic optimization (see Section *10.2.1*), except for the addition of dual constraints

$$\left(\overline{C}_j - \sum_{i=0}^{m} y_i \overline{A}_{ij}\right) \in \mathcal{S}_+^{r_j}.$$

Note that the dual of the dual problem is identical to the original primal problem.

10.3.2 Infeasibility for Semidefinite Optimization

In case **MOSEK** finds a problem to be infeasible it reports a certificate of the infeasibility. This works exactly as for linear problems (see Section *10.1.2*).

Primal Infeasible Problems

If the problem (10.9) is infeasible, **MOSEK** will report a certificate of primal infeasibility: The dual solution reported is a certificate of infeasibility, and the primal solution is undefined.

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

$$\begin{array}{lll}
\text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x & \\
\text{subject to} & & \\
& A^T y + s_l^x - s_u^x + s_n^x = 0, & \\
& \sum_{i=0}^{m-1} y_i \overline{A}_{ij} + \overline{S}_j = 0, & j = 0, \ldots, p-1 \\
& -y + s_l^c - s_u^c = 0, & \\
& s_l^c, s_u^c, s_l^x, s_u^x \geq 0, & \\
& s_n^x \in \mathcal{K}^*, \quad \overline{S}_j \in \mathcal{S}_+^{r_j}, & j = 0, \ldots, p-1
\end{array}$$

such that the objective value is strictly positive.

Dual Infeasible Problems

If the problem (10.10) is infeasible, **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 problem

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

where

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

and

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

such that the objective value is strictly negative.

www.EngineeringBooksPdf.com

10.4 Quadratic and Quadratically Constrained Optimization

A convex quadratic and quadratically constrained optimization problem is an optimization problem of the form

$$\begin{array}{lccccll} \text{minimize} & & & \frac{1}{2}x^T Q^o x + c^T x + c^f & & & \\ \text{subject to} & l_k^c & \leq & \frac{1}{2}x^T Q^k x + \sum_{j=0}^{n-1} a_{kj} x_j & \leq & u_k^c, & k = 0, \ldots, m-1, \\ & l_j^x & \leq & x_j & \leq & u_j^x, & j = 0, \ldots, n-1, \end{array} \tag{10.11}$$

where 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 fullfiled.

10.4.1 A Recommendation

Any convex quadratic optimization problem can be reformulated as a conic quadratic optimization problem, see *[MOSEKApS12]* and in particular *[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 formuation if deemed worthwhile potentially leading to (huge) computational savings.

However, instead of relying on the automatic reformulation we recommend to formulate the problem as 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 neeeded for conic problems.
- usually the modeller can do a better reformulation than the automatic method because the modeller can exploit the knowledge of what is being modelled.

To summarize we recommend to formulate quadratic problems and in particular quadratically constrained problems directly in conic form.

10.4.2 Duality for Quadratic and Quadratically Constrained Optimization

The dual problem corresponding to the quadratic and quadratically constrained optimization problem (10.11) is given by

$$\begin{array}{ll} \text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x + \frac{1}{2}x^T \left\{ \sum_{k=0}^{m-1} y_k Q^k - Q^o \right\} x + c^f \\ \text{subject to} & \\ & A^T y + s_l^x - s_u^x + \left\{ \sum_{k=0}^{m-1} y_k Q^k - Q^o \right\} x = c, \\ & -y + s_l^c - s_u^c = 0, \\ & s_l^c, s_u^c, s_l^x, s_u^x \geq 0. \end{array} \tag{10.12}$$

The dual problem is related to the dual problem for linear optimization (see Section *10.1.1*), 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 (10.11) and the dual problem (10.12).

www.EngineeringBooksPdf.com

10.4.3 Infeasibility for Quadratic and Quadratically Constrained Optimization

In case **MOSEK** finds a problem to be infeasible it reports a certificate of infeasibility. This works exactly as for linear problems (see Section *10.1.2*).

Primal Infeasible Problems

If the problem (10.11) with all $Q^k = 0$ is infeasible, **MOSEK** will report a certificate of primal infeasibility. As the constraints are the same as for a linear problem, the certificate of infeasibility is the same as for linear optimization (see Section *10.1.2.1*).

Dual Infeasible Problems

If the problem (10.12) with all $Q^k = 0$ is infeasible, **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 problem

$$\begin{array}{lccccc} \text{minimize} & & & c^T x & & \\ \text{subject to} & \hat{l}^c & \leq & Ax & \leq & \hat{u}^c, \\ & 0 & \leq & Q^o x & \leq & 0, \\ & \hat{l}^x & \leq & x & \leq & \hat{u}^x, \end{array}$$

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 the objective value is strictly negative.

10.5 General Convex Optimization

MOSEK is capable of solving smooth (twice differentiable) convex nonlinear optimization problems of the form

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

where

- m is the number of constraints.
- n is the number of decision variables.
- $x \in \mathbb{R}^n$ is a vector of decision variables.
- $c \in \mathbb{R}^n$ is the linear part objective function.
- $A \in \mathbb{R}^{m \times n}$ is the constraint matrix.
- $l^c \in \mathbb{R}^m$ is the lower limit on the activity for the constraints.
- $u^c \in \mathbb{R}^m$ is the upper limit on the activity for the constraints.
- $l^x \in \mathbb{R}^n$ is the lower limit on the activity for the variables.

www.EngineeringBooksPdf.com

- $u^x \in \mathbb{R}^n$ is the upper limit on the activity for the variables.
- $f : \mathbb{R}^n \to \mathbb{R}$ is a nonlinear function.
- $g : \mathbb{R}^n \to \mathbb{R}^m$ is a nonlinear vector function.

This means that the i-th constraint has the form

$$l_i^c \leq g_i(x) + \sum_{j=1}^{n} a_{ij} x_j \leq u_i^c.$$

The linear term Ax is included in $g(x)$ since it can be handled much more efficiently as a separate entity when optimizing.

The nonlinear functions f and g must be smooth in all $x \in [l^x; u^x]$. Moreover, $f(x)$ must be a convex function and $g_i(x)$ must satisfy

$$\begin{array}{rcl} -\infty < l_i^c & \Rightarrow & g_i(x) \text{ is concave,} \\ u_i^c < \infty & \Rightarrow & g_i(x) \text{ is convex,} \\ -\infty < l_i^c \leq u_i^c < \infty & \Rightarrow & g_i(x) = 0. \end{array}$$

10.5.1 Duality for General convex Optimization

Similar to the linear case, **MOSEK** reports dual information in the general nonlinear case. Indeed in this case the Lagrange function is defined by

$$\begin{array}{rcl} L(x, s_l^c, s_u^c, s_l^x, s_u^x) & := & f(x) + c^T x + c^f \\ & & -(s_l^c)^T (g(x) + Ax - l^c) - (s_u^c)^T (u^c - g(x) - Ax) \\ & & -(s_l^x)^T (x - l^x) - (s_u^x)^T (u^x - x), \end{array}$$

and the dual problem is given by

$$\begin{array}{lcl} \text{maximize} & L(x, s_l^c, s_u^c, s_l^x, s_u^x) & \\ \text{subject to} & \nabla_x L(x, s_l^c, s_u^c, s_l^x, s_u^x)^T & = \quad 0, \\ & s_l^c, s_u^c, s_l^x, s_u^x \geq 0, & \end{array}$$

which is equivalent to

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

In this context we use the following definition for scalar functions

$$\nabla f(x) \quad = \quad \left[\frac{\partial f(x)}{\partial x_1}, \ldots, \frac{\partial f(x)}{\partial x_n} \right],$$

and accordingly for vector functions

$$\nabla g(x) \quad = \quad \begin{bmatrix} \nabla g_1(x) \\ \vdots \\ \nabla g_m(x) \end{bmatrix}.$$

www.EngineeringBooksPdf.com

CHAPTER

ELEVEN

THE OPTIMIZERS FOR CONTINUOUS PROBLEMS

The most essential part of **MOSEK** is the optimizers. Each optimizer is designed to solve a particular class of problems, i.e. linear, conic, or general nonlinear problems. The purpose of the present chapter is to discuss which optimizers are available for the continuous problem classes and how the performance of an optimizer can be tuned, if needed. This chapter deals with the optimizers for *continuous problems* with no integer variables.

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

1. *Presolve*: Preprocessing to reduce the size of the problem.
2. *Dualizer*: Choosing whether to solve the primal or the dual form of the problem.
3. *Scaling*: Scaling the problem for better numerical stability.
4. *Optimize*: Solve the problem using selected method.

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

Using multiple threads

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

By default **MOSEK** will automatically select the number of threads to be employed when solving the problem. However, the number of threads employed can be changed by setting the parameter *iparam.num_threads*. This should never exceed the number of cores on the computer.

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

For small problems, using multiple threads is not be worthwhile and may even be counter productive.

11.1 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

www.EngineeringBooksPdf.com

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 then it is suggested 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 presolve off may help.

Since all computations are done in finite prescision then the presolve employs some tolerances when concluding a variable is fixed or 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.

www.EngineeringBooksPdf.com

It is best practise to build models without linear dependencies. If the linear dependencies are removed at the modeling stage, the linear dependency check can safely be disabled by setting the parameter *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 most efficient to solve the primal or dual problem. The form (primal or dual) solved is displayed in the **MOSEK** log. Should the internal heuristics not choose the most efficient form of the problem it may be worthwhile to set the dualizer manually by setting the parameters:

- *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 problems may be 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 calculations. Since computers work in finite precision, extreme coefficients should be avoided. In general, data around the same *order of magnitude* is preferred, and we will refer to a problem, satisfying this loose property, as being *well-scaled*. If the problem is not well scaled, **MOSEK** will try to scale (multiply) constraints and variables by suitable constants. **MOSEK** solves the scaled problem to improve the numerical properties.

The scaling process is transparent, i.e. the solution to the original problem is reported. It is important to be aware that the optimizer terminates when the termination criterion is met on the scaled problem, therefore significant primal or dual infeasibilities may occur after unscaling for badly scaled problems. The best solution 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.

11.2 Linear Optimization

11.2.1 Optimizer Selection

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

11.2.2 The Interior-point Optimizer

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

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

$$\begin{array}{lll} \text{minimize} & c^T x & \\ \text{subject to} & Ax & = b, \\ & x \geq 0. & \end{array} \tag{11.1}$$

www.EngineeringBooksPdf.com

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 (11.1) has an optimal solution, is primal infeasible or is dual infeasible, the optimization algorithm must deal with all three situations. This is the reason that **MOSEK** solves the so-called homogeneous model

$$\begin{array}{rcl} Ax - b\tau & = & 0, \\ A^T y + s - c\tau & = & 0, \\ -c^T x + b^T y - \kappa & = & 0, \\ x, s, \tau, \kappa & \geq & 0, \end{array} \tag{11.2}$$

where y and s correspond to the dual variables in (11.1), and τ and κ are two additional scalar variables. Note that the homogeneous model (11.2) 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 (11.2) 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.

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

$$-c^T x^* > 0 \tag{11.3}$$

or

$$b^T y^* > 0 \tag{11.4}$$

is satisfied. If (11.3) is satisfied then x^* is a certificate of dual infeasibility, whereas if (11.4) is satisfied then y^* is a certificate of dual infeasibility.

In summary, by computing an appropriate solution to the homogeneous model, all information required for a solution to the original problem is obtained. A solution to the homogeneous model can be computed using a primal-dual interior-point algorithm *[And09]*.

www.EngineeringBooksPdf.com

Interior-point Termination Criterion

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

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

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

to homogeneous model is generated where

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

Whenever the trial solution satisfies the criterion

$$\begin{array}{rcl} \left\| A\frac{x^k}{\tau^k} - b \right\|_\infty & \leq & \epsilon_p(1 + \|b\|_\infty), \\ \left\| A^T\frac{y^k}{\tau^k} + \frac{s^k}{\tau^k} - c \right\|_\infty & \leq & \epsilon_d(1 + \|c\|_\infty), \text{ and} \\ \min\left(\frac{(x^k)^T s^k}{(\tau^k)^2}, |\frac{c^T x^k}{\tau^k} - \frac{b^T y^k}{\tau^k}|\right) & \leq & \epsilon_g \max\left(1, \frac{\min(|c^T x^k|, |b^T y^k|)}{\tau^k}\right), \end{array} \tag{11.5}$$

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 (11.5) 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.

On the other hand, if the trial solution satisfies

$$-\epsilon_i c^T x^k > \frac{\|c\|_\infty}{\max(1, \|b\|_\infty)} \left\|Ax^k\right\|_\infty$$

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

$$\left\|Ax^k\right\|_\infty > 0,$$

and define

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

It is easy to verify that

$$\|A\bar{x}\|_\infty = \epsilon_i \frac{\max(1, \|b\|_\infty)}{\|c\|_\infty} \text{ and } -c^T\bar{x} > 1,$$

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

Finally, if

$$\epsilon_i b^T y^k > \frac{\|b\|_\infty}{\max(1, \|c\|_\infty)} \left\|A^T y^k + s^k\right\|_\infty$$

www.EngineeringBooksPdf.com

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

It is possible to adjust the tolerances ε_p, ε_d, ε_g and ε_i using parameters; see Table 11.1 for details.

Table 11.1: 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 (11.5) reveals that quality of the solution is dependent 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 and ε_g, have to be relaxed together to achieve an effect.

In some cases the interior-point method terminates having found a solution not too far from meeting the optimality condition (11.5). A solution is defined as *near optimal* if scaling ε_p, ε_d and ε_g by any number $\varepsilon_n \in [1.0, +\infty]$ conditions (11.5) are satisfied.

A near optimal solution is therefore of lower quality but still potentially valuable. If for instance the solver stalls, i.e. it can make no more significant progress towards the optimal solution, a near optimal solution could be available and be good enough for the user.

The basis identification discussed in Section *11.2.2.2* requires an optimal solution to work well; hence basis identification should be turned off if the termination criterion is relaxed.

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

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 relaxation 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:

$$\begin{array}{lrcl} \text{minimize} & x+y & & \\ \text{subject to} & x+y & = & 1, \\ & x,y \geq 0. & & \end{array}$$

It is easy to see that all feasible solutions are also optimal. In particular, there are two basic solutions namely

$$\begin{array}{rcl} (x_1^*, y_1^*) & = & (1,0), \\ (x_2^*, y_2^*) & = & (0,1). \end{array}$$

www.EngineeringBooksPdf.com

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* phase be short. However, in some cases for nasty problems e.g. 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 can be tuned by the *iparam.bi_clean_optimizer* parameter i.e. primal or dual simplex, and the maximum number of iterations can be set by the *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 presented:

```
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 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 Section *11.2.2* the columns of the iteration log have the following meaning:

- `ITE`: Iteration index.
- `PFEAS`: $\left\|Ax^k - b\tau^k\right\|_\infty$. The numbers in this column should converge monotonically towards to 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 toward to zero but may stall at low level due to rounding errors.

www.EngineeringBooksPdf.com

- `GFEAS`: $|-c^T x^k + b^T y^k - \kappa^k|$. The numbers in this column should converge monotonically toward to 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 monotonically to zero.
- `TIME`: Time spend since the optimization started.

11.2.3 The simplex Based Optimizer

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

The simplex optimizer uses a different method that allows exploiting an initial guess for the optimal solution to reduce the solution time. Depending on the problem it may be faster or slower to use an initial guess; see section *11.2.4* for a discussion.

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

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 Section *10.1* and *10.1.1* for a definition of the primal and dual problem. Due to the fact that computations are performed in finite precision **MOSEK** allows violation of primal and dual feasibility within certain tolerances. The user can control the allowed primal and dual tolerances with the parameters *dparam.basis_tol_x* and *dparam.basis_tol_s*.

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.

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.

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** counts a "numerical unexpected behavior" event inside the optimizer as a *set-back*. The user can define how many set-backs the optimizer accepts; if that number is exceeded, the optimization will be aborted. Set-backs are implemented to avoid long sequences where the optimizer tries to recover from an unstable situation.

Set-backs are, for example, repeated singularities when factorizing the basis matrix, repeated loss of feasibility, degeneracy problems (no progress in objective) and other events indicating numerical difficulties. If the simplex optimizer encounters a lot of set-backs the problem is usually badly scaled; in such

www.EngineeringBooksPdf.com

a situation try to reformulate into a better scaled problem. Then, if a lot of set-backs still occur, trying one or more of the following suggestions may be worthwhile:

- Raise tolerances for allowed primal or dual feasibility: Hence, increase the value of
  - *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 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.

11.2.4 The Interior-point or the Simplex Optimizer?

Given a linear optimization problem, which optimizer is the best: The primal simplex, the dual simplex or the interior-point optimizer?

It is impossible to provide a general answer to this question. However, the interior-point optimizer behaves more predictably: it tends to use between 20 and 100 iterations, almost independently of problem size, but cannot perform warm-start, while simplex can take advantage of an initial solution, but is less predictable for cold-start. The interior-point optimizer is used by default.

11.2.5 The Primal or the Dual Simplex Variant?

MOSEK provides both a primal and a dual simplex optimizer. Predicting which simplex optimizer is faster is impossible, however, in recent years the dual optimizer has seen several algorithmic and computational improvements, which, in our experience, makes it faster on average than the primal simplex optimizer. Still, it depends much on the problem structure and size.

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

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

11.3 Conic Optimization

11.3.1 The Interior-point Optimizer

For conic optimization problems only an interior-point type optimizer is available. The interior-point optimizer is an implementation of the so-called homogeneous and self-dual algorithm. For a detailed description of the algorithm, please see *[ART03]*.

www.EngineeringBooksPdf.com

Interior-point Termination Criteria

The parameters controlling when the conic interior-point optimizer terminates are shown in Table 11.2.

Table 11.2: Parameters employed in termination criterion.

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

11.4 Nonlinear Convex Optimization

11.4.1 The Interior-point Optimizer

For quadratic, quadratically constrained, and general convex optimization problems an interior-point type optimizer is available. The interior-point optimizer is an implementation of the homogeneous and self-dual algorithm. For a detailed description of the algorithm, please see *[AY98]*, *[AY99]*.

The Convexity Requirement

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

The following parameters are available to control the convexity check:

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

The Differentiabilty Requirement

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

For instance, the function

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

is differentiable everywhere whereas the function

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

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

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

$$x \geq 0$$

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

www.EngineeringBooksPdf.com

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

$$f(x) = \frac{1}{x} + \frac{1}{1-x}. \tag{11.6}$$

In this case the bounds

$$0 \leq x \leq 1$$

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

For efficiency reasons it may be better to reformulate the problem than to force **MOSEK** to observe ranged bounds strictly. For instance, (11.6) can be reformulated as follows

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

Interior-point Termination Criteria

The parameters controlling when the general convex interior-point optimizer terminates are shown in Table 11.3.

Table 11.3: Parameters employed in termination criteria.

Parameter name	Purpose
dparam.intpnt_nl_tol_pfeas	Controls primal feasibility
dparam.intpnt_nl_tol_dfeas	Controls dual feasibility
dparam.intpnt_nl_tol_rel_gap	Controls relative gap
dparam.intpnt_tol_infeas	Controls when the problem is declared infeasible
dparam.intpnt_nl_tol_mu_red	Controls when the complementarity is reduced enough

11.5 Using Multiple Threads in an Optimizer

If multiple cores are available then it is possible for **MOSEK** to take advantage of them to speed up the computation. However, please note the speedup achieved is going to be dependent on the problem characteristics e.g. the size of problem. Typically for smallish problems no speedup is obtained by exploiting multiple cores. In fact forcing **MOSEK** to use one core can increase speed because parallel overhead is avoided.

11.5.1 Thread Safety

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

11.5.2 Determinism

The optimizers are run-to-run deterministic which means if a problem is solved twice on the same computer using the same parameter setting and exactly the same input then exactly the same results is obtained. One qualification is that no time limits must be imposed because the time taken to perform an operation on a computer is dependent on many factors such as the current workload.

www.EngineeringBooksPdf.com

11.5.3 The Parallelized Interior-point Optimizer

By default the interior-point optimizer exploits multiple cores using multithreading. Hence, big tasks such as large dense matrix multiplication may be divided into several independent smaller tasks that can be computed independently. However, there is a computational overhead associated with exploiting multiple threads e.g. cost of the additional coordination etc. Therefore, it may be advantageous to turn off the mutithreading for smallish problem using the parameter *iparam.intpnt_multi_thread*.

Moreover, when the interior-point optimizer is allowed to exploit multiple threads, then the parameter *iparam.num_threads* controls the maximum number of threads (and therefore the number of cores) that **MOSEK** will employ.

www.EngineeringBooksPdf.com

CHAPTER

TWELVE

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. **MOSEK** can solve mixed-integer

- linear,
- quadratic and quadratically constrained, and
- conic qudratic

problems.

Readers unfamiliar with integer optimization are recommended to consult some relevant literature, e.g. the book *[Wol98]* by Wolsey.

12.1 Some Concepts and Facts Related to Mixed-integer Optimization

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

The mixed-integer optimization problem

$$\begin{array}{rcllll} z^* & = & \text{minimize} & c^T x & & \\ & & \text{subject to} & Ax & = & b, \\ & & & x \geq 0 & & \\ & & & x_j \in \mathbb{Z}, & & \forall j \in \mathcal{J}, \end{array} \tag{12.1}$$

has the continuous relaxation

$$\begin{array}{rcllll} \underline{z} & = & \text{minimize} & c^T x & & \\ & & \text{subject to} & Ax & = & b, \\ & & & x \geq 0 & & \end{array} \tag{12.2}$$

The continuos relaxation is identical to the mixed-integer problem with the restriction that some variables must be integers removed.

There are two important observations about the continuous relaxation. First, the continuous relaxation is usually much faster to optimize than the mixed-integer problem. Secondly if $\hat{x}$ is any feasible solution to (12.1) and

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

www.EngineeringBooksPdf.com

then

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

This is an important observation since if it is only possible to find a near optimal solution within a reasonable time frame then the quality of the solution can nevertheless be evaluated. The value $\underline{z}$ is a lower bound on the optimal objective value. This implies that the obtained solution 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.

12.2 The Mixed-integer Optimizer

The mixed-integer optimizer can handle problems with linear, quadratic objective and constraints and conic constraints. However, a problem can not contain both quadratic objective or constraints and conic constraints.

The mixed-integer optimizer is specialized for solving linear and conic optimization problems. It can also solve pure quadratic and quadratically constrained problems; these problems are automatically converted to conic problems before being solved.

The mixed-integer optimizer is run-to-run deterministic. This means that if a problem is solved twice on the same computer with identical options then the obtained solution will be bit-for-bit identical for the two runs. However, 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:** In this phase the optimizer tries to reduce the size of the problem and improve the formulation using preprocessing techniques. The presolve stage can be turned off using the *iparam.presolve_use* parameter
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

12.3 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. Therefore, 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

$$\bar{z} - \underline{z} \leq \max(\delta_2, \delta_3 \max(10^{-10}, |\bar{z}|))$$

www.EngineeringBooksPdf.com

is satisfied. If this is the case, the integer optimizer terminates and reports the integer feasible solution as an optimal solution. Please note that $\underline{z}$ is a valid lower bound determined by the integer optimizer during the solution process, i.e.

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

The lower bound $\underline{z}$ normally increases during the solution process.

12.3.1 Relaxed Termination

If an optimal solution cannot be located within a reasonable time, it may be advantageous to employ a relaxed termination criterion after some time. Whenever the integer optimizer locates an integer feasible solution and has spent at least the number of seconds defined by the *dparam.mio_disable_term_time* parameter on solving the problem, it will check whether the criterion

$$\bar{z} - \underline{z} \leq \max(\delta_4, \delta_5 \max(10^{-10}, |\bar{z}|))$$

is satisfied. If it is satisfied, the optimizer will report that the candidate solution is **near optimal** and then terminate. Please note that since this criterion depends on timing, the optimizer will not be run to run deterministic.

12.4 Parameters Affecting the Termination of the Integer Optimizer.

All δ tolerances can be adjusted using suitable parameters — see Table 12.1.

Table 12.1: 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_near_tol_abs_gap*
δ_5	*dparam.mio_near_tol_rel_gap*

In Table 12.2 some other parameters affecting the integer optimizer termination criterion are shown. Please note that if the effect of a parameter is delayed, the associated termination criterion is applied only after some time, specified by the *dparam.mio_disable_term_time* parameter.

Table 12.2: Other parameters affecting the integer optimizer termination criterion.

Parameter name	Delayed	Explanation
iparam.mio_max_num_branches	Yes	Maximum number of branches allowed.
iparam.mio_max_num_relaxs	Yes	Maximum number of relaxations allowed.
iparam.mio_max_num_solutions	Yes	Maximum number of feasible integer solutions allowed.

12.5 How to Speed Up the Solution Process

As mentioned previously, in many cases it is not possible to find an optimal solution to an integer optimization problem in a reasonable amount of time. Some suggestions to reduce the solution time are:

www.EngineeringBooksPdf.com

- Relax the termination criterion: In case the run time is not acceptable, the first thing to do is to relax the termination criterion — see Section *12.3* for details.
- Specify a good initial solution: In many cases a good feasible solution is either known or easily computed using problem specific knowledge. If a good feasible solution is known, it is usually worthwhile to use this as a starting point for the integer optimizer.
- Improve the formulation: A mixed-integer optimization problem may be impossible to solve in one form and quite easy in another form. However, it is beyond the scope of this manual to discuss good formulations for mixed-integer problems. For discussions on this topic see for example *[Wol98]*.

12.6 Understanding Solution Quality

To determine the quality of the solution one should check the following:

- The solution status key returned by **MOSEK**
- The *optimality gap*: A measure of how much the located solution can deviate from the optimal solution to the problem
- Feasibility. How much the solution violates the constraints of the problem

The *optimality gap* is a measure for how close the solution is to the optimal solution. The optimality gap is given by

$$\epsilon = |(\text{objective value of feasible solution}) - (\text{objective bound})|.$$

The objective value of the solution is guarantied to be within ϵ of the optimal solution.

The optimality gap can be retrieved through the solution item *dinfitem.mio_obj_abs_gap*. Often it is more meaningful to look at the optimality gap normalized with the magnitude of the solution. The relative optimality gap is available in *dinfitem.mio_obj_rel_gap*.

www.EngineeringBooksPdf.com

CHAPTER

THIRTEEN

PROBLEM ANALYZER

The problem analyzer prints a detailed survey of the

- linear constraints and objective
- quadratic constraints
- conic constraints
- variables

of the model.

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.

```
Analyzing the problem

Constraints               Bounds                    Variables
 upper bd:       421       ranged  : all             cont:       421
 fixed   :        58                                 bin :       421

--------------------------------------------------------------------------------

Objective, min cx
   range: min |c|: 0.00000   min |c|>0: 11.0000     max |c|: 500.000
 distrib:        |c|        vars
                   0         421
           [11, 100)         150
          [100, 500]         271

--------------------------------------------------------------------------------

Constraint matrix A has
       479 rows (constraints)
       842 columns (variables)
      2091 (0.518449%) nonzero entries (coefficients)

Row nonzeros, A_i
   range: min A_i: 2 (0.23753%)    max A_i: 34 (4.038%)
 distrib:        A_i        rows       rows%        acc%
                   2         421       87.89       87.89
             [8, 15]          20        4.18       92.07
            [16, 31]          30        6.26       98.33
            [32, 34]           8        1.67      100.00

Column nonzeros, A|j
   range: min A|j: 2 (0.417537%)    max A|j: 3 (0.626305%)
 distrib:        A|j        cols       cols%        acc%
                   2         435       51.66       51.66
```


www.EngineeringBooksPdf.com

```
                  3         407       48.34      100.00

A nonzeros, A(ij)
   range: min |A(ij)|: 1.00000     max |A(ij)|: 100.000
 distrib:      A(ij)      coeffs
             [1, 10)        1670
           [10, 100]         421

--------------------------------------------------------------------------------

Constraint bounds, lb <= Ax <= ub
 distrib:        |b|             lbs             ubs
                   0                             421
             [1, 10]              58              58

Variable bounds, lb <= x <= ub
 distrib:        |b|             lbs             ubs
                   0             842
             [1, 10)                             421
           [10, 100]                             421

--------------------------------------------------------------------------------
```

The survey is divided into six different sections, each described below. To keep the presentation short with focus on key elements the analyzer generally attempts to display information on issues relevant for the current model only: E.g., if the model does not have any conic constraints (this is the case in the example above) or any integer variables, those parts of the analysis will not appear.

13.1 General Characteristics

The first part of the survey consists of a brief summary of the model's linear and quadratic constraints (indexed by i) and variables (indexed by j). The summary is divided into three subsections:

Constraints

- `upper bd` The number of upper bounded constraints, $\sum_{j=0}^{n-1} a_{ij}x_j \leq u_i^c$
- `lower bd` The number of lower bounded constraints, $l_i^c \leq \sum_{j=0}^{n-1} a_{ij}x_j$
- `ranged` The number of ranged constraints, $l_i^c \leq \sum_{j=0}^{n-1} a_{ij}x_j \leq u_i^c$
- `fixed` The number of fixed constraints, $l_i^c = \sum_{j=0}^{n-1} a_{ij}x_j = u_i^c$
- `free` The number of free constraints

Bounds

- `upper bd` The number of upper bounded variables, $x_j \leq u_j^x$
- `lower bd` The number of lower bounded variables, $l_k^x \leq x_j$
- `ranged` The number of ranged variables, $l_k^x \leq x_j \leq u_j^x$
- `fixed` The number of fixed variables, $l_k^x = x_j = u_j^x$
- `free` The number of free variables

www.EngineeringBooksPdf.com

`Variables`

- `cont` The number of continuous variables, $x_j \in \mathbb{R}$
- `bin` The number of binary variables, $x_j \in \{0, 1\}$
- `int` The number of general integer variables, $x_j \in \mathbb{Z}$

Only constraints, bounds and domains actually in the model will be reported on; if all entities in a section turn out to be of the same kind, the number will be replaced by `all` for brevity.

13.2 Objective

The second part of the survey focuses on (the linear part of) the objective, summarizing the optimization sense and the coefficients' absolute value range and distribution. The number of 0 (zero) coefficients is singled out (if any such variables are in the problem).

The range is displayed using three terms:

- `min |c|` The minimum absolute value among all coeffecients
- `min |c|>0` The minimum absolute value among the nonzero coefficients
- `max |c|` The maximum absolute value among the coefficients

If some of these extrema turn out to be equal, the display is shortened accordingly:

- If `min |c|` is greater than zero, the `min |c|>0` term is obsolete and will not be displayed
- If only one or two different coefficients occur this will be displayed using `all` and an explicit listing of the coefficients

The absolute value distribution is displayed as a table summarizing the numbers by orders of magnitude (with a ratio of 10). Again, the number of variables with a coefficient of 0 (if any) is singled out. Each line of the table is headed by an interval (half-open intervals including their lower bounds), and is followed by the number of variables with their objective coefficient in this interval. Intervals with no elements are skipped.

13.3 Linear Constraints

The third part of the survey displays information on the nonzero coefficients of the linear constraint matrix.

Following a brief summary of the matrix dimensions and the number of nonzero coefficients in total, three sections provide further details on how the nonzero coefficients are distributed by row-wise count (`A_i`), by column-wise count (`A|j`), and by absolute value (`|A(ij)|`). Each section is headed by a brief display of the distribution's range (`min` and `max`), and for the row/column-wise counts the corresponding densities are displayed too (in parentheses).

The distribution tables single out three particularly interesting counts: zero, one, and two nonzeros per row/column; the remaining row/column nonzeros are displayed by orders of magnitude (ratio 2). For each interval the relative and accumulated relative counts are also displayed.

Note that constraints may have both linear and quadratic terms, but the empty rows and columns reported in this part of the survey relate to the linear terms only. If empty rows and/or columns are found in the linear constraint matrix, the problem is analyzed further in order to determine if the corresponding constraints have any quadratic terms or the corresponding variables are used in conic or quadratic constraints.

The distribution of the absolute values, `|A(ij)|`, is displayed just as for the objective coefficients described above.

www.EngineeringBooksPdf.com

13.4 Constraint and Variable Bounds

The fourth part of the survey displays distributions for the absolute values of the finite lower and upper bounds for both constraints and variables. The number of bounds at 0 is singled out and, otherwise, displayed by orders of magnitude (with a ratio of 10).

13.5 Quadratic Constraints

The fifth part of the survey displays distributions for the nonzero elements in the gradient of the quadratic constraints, i.e. the nonzero row counts for the column vectors Qx . The table is similar to the tables for the linear constraints' nonzero row and column counts described in the survey's third part.

Note: Quadratic constraints may also have a linear part, but that will be included in the linear constraints survey; this means that if a problem has one or more pure quadratic constraints, part three of the survey will report anleq al number of linear constraint rows with 0 (zero) nonzeros. Likewise, variables that appear in quadratic terms only will be reported as empty columns (0 nonzeros) in the linear constraint report.

13.6 Conic Constraints

The last part of the survey summarizes the model's conic constraints. For each of the two types of cones, quadratic and rotated quadratic, the total number of cones are reported, and the distribution of the cones' dimensions are displayed using intervals. Cone dimensions of 2, 3, and 4 are singled out.

www.EngineeringBooksPdf.com

CHAPTER

FOURTEEN

ANALYZING INFEASIBLE PROBLEMS

When developing and implementing a new optimization model, the first attempts will often be either infeasible, due to specification of inconsistent constraints, or unbounded, if important constraints have been left out.

In this section we will

- go over an example demonstrating how to locate infeasible constraints using the **MOSEK** infeasibility report tool,
- discuss in more general terms which properties may cause infeasibilities, and
- present the more formal theory of infeasible and unbounded problems.

14.1 Example: Primal Infeasibility

A problem is said to be *primal infeasible* if no solution exists that satisfies all the constraints of the problem.

As an example of a primal infeasible problem consider the problem of minimizing the cost of transportation between a number of production plants and stores: Each plant produces a fixed number of goods, and each store has a fixed demand that must be met. Supply, demand and cost of transportation per unit are given in Fig. 14.1.

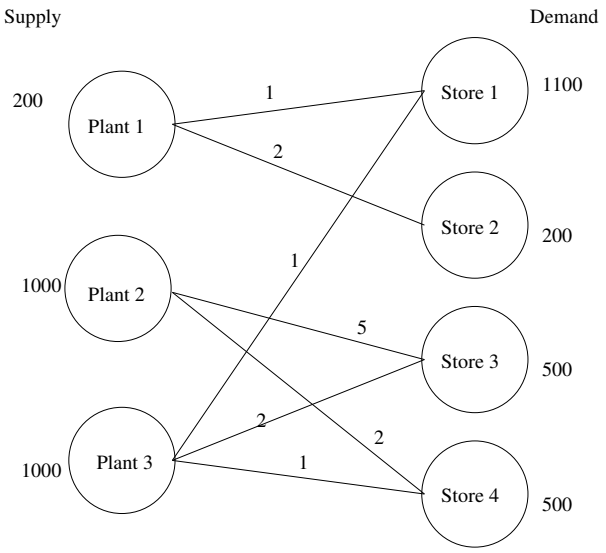


Fig. 14.1: Supply, demand and cost of transportation.

The problem represented in Fig. 14.1 is infeasible, since the total demand

$$2300 = 1100 + 200 + 500 + 500$$

www.EngineeringBooksPdf.com

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:

$$\begin{array}{lrcrcrcrcrcrcrl}
\text{minimize} & x_{11} & + & 2x_{12} & + & 5x_{23} & + & 2x_{24} & + & x_{31} & + & 2x_{33} & + & x_{34} & \\
\text{subject to} & x_{11} & + & x_{12} & & & & & & & & & & & \leq \ 200, \\
 & & & & & x_{23} & + & x_{24} & & & & & & & \leq \ 1000, \\
 & & & & & & & & & x_{31} & + & x_{33} & + & x_{34} & \leq \ 1000, \\
 & x_{11} & & & & & & & + & x_{31} & & & & & = \ 1100, \\
 & & & x_{12} & & & & & & & & & & & = \ 200, \\
 & & & & & x_{23} & + & & & & & x_{33} & & & = \ 500, \\
 & & & & & & & x_{24} & + & & & & & x_{34} & = \ 500, \\
x_{ij} \geq 0. & & & & & & & & & & & & & &
\end{array} \tag{14.1}$$

Solving problem (14.1) using **MOSEK** will result in a solution, a solution status and a problem status. Among the log output from the execution of **MOSEK** on the above problem are the lines:

```
Basic solution
Problem status  : PRIMAL_INFEASIBLE
Solution status : PRIMAL_INFEASIBLE_CER
```

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

14.2 Locating the cause of Primal Infeasibility

Usually a primal infeasible problem status is caused by a mistake in formulating the problem and therefore the question arises: *What is the cause of the infeasible status?* When trying to answer this question, it is often advantageous to follow these steps:

- Remove the objective function. This does not change the infeasibility status but simplifies the problem, eliminating any possibility of issues related to the objective function.
- Consider whether your problem has some necessary conditions for feasibility and examine if these are satisfied, e.g. total supply should be greater than or equal to total demand.
- Verify that coefficients and bounds are reasonably sized in your problem.

If the problem is still primal infeasible, some of the constraints must be relaxed or removed completely. The **MOSEK** infeasibility report (Section *14.4*) may assist you in finding the constraints causing the infeasibility.

Possible ways of relaxing your problem nclude:

- Increasing (decreasing) upper (lower) bounds on variables and constraints.
- Removing suspected constraints from the problem.

Returning to the transportation example, we discover that removing the fifth constraint

$$x_{12} = 200$$

makes the problem feasible.

www.EngineeringBooksPdf.com

14.3 Locating the Cause of Dual Infeasibility

A problem may also be *dual infeasible*. In this case the primal problem is often unbounded, meaning that feasbile solutions exists such that the objective tends towards infinity. An example of a dual infeasible and primal unbounded problem is:

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

To resolve a dual infeasibility the primal problem must be made more restricted by

- Adding upper or lower bounds on variables or constraints.
- Removing variables.
- Changing the objective.

14.3.1 A cautionary note

The problem

$$\begin{array}{ll} \text{minimize} & 0 \\ \text{subject to} & 0 \leq x_1, \\ & x_j \leq x_{j+1}, \quad j = 1, \ldots, n-1, \\ & x_n \leq -1 \end{array}$$

is clearly infeasible. Moreover, if any one of the constraints is dropped, then the problem becomes feasible.

This illustrates the worst case scenario where all, or at least a significant portion of the constraints are involved in causing infeasibility. Hence, it may not always be easy or possible to pinpoint a few constraints responsible for infeasibility.

14.4 The Infeasibility Report

MOSEK includes functionality for diagnosing the cause of a primal or a dual infeasibility. It can be turned on by setting the *iparam.infeas_report_auto* to *onoffkey.on*. This causes **MOSEK** to print a report on variables and constraints involved in the infeasibility.

The *iparam.infeas_report_level* parameter controls the amount of information presented in the infeasibility report. The default value is 1.

14.4.1 Example: Primal Infeasibility

We will keep working with the problem (14.1) written in LP format:

Listing 14.1: The code for problem (14.1).

```
\
\ An example of an infeasible linear problem.
\
minimize
 obj: + 1 x11 + 2 x12
      + 5 x23 + 2 x24
      + 1 x31 + 2 x33 + 1 x34
st
  s0: + x11 + x12       <= 200
  s1: + x23 + x24       <= 1000
  s2: + x31 + x33 + x34 <= 1000
```


www.EngineeringBooksPdf.com

```
  d1: + x11 + x31         = 1100
  d2: + x12               = 200
  d3: + x23 + x33         = 500
  d4: + x24 + x34         = 500
bounds
end
```

14.4.2 Example: Dual Infeasibility

The following problem is dual to (14.1) and therefore it is dual infeasible.

Listing 14.2: The dual of problem (14.1).

```
maximize + 200 y1 + 1000 y2 + 1000 y3 + 1100 y4 + 200 y5 + 500 y6 + 500 y7
subject to
   x11: y1+y4 < 1
   x12: y1+y5 < 2
   x23: y2+y6 < 5
   x24: y2+y7 < 2
   x31: y3+y4 < 1
   x33: y3+y6 < 2
   x34: y3+y7 < 1
bounds
   -inf <= y1 < 0
   -inf <= y2 < 0
   -inf <= y3 < 0
   y4 free
   y5 free
   y6 free
   y7 free
end
```

This can be verified by proving that

$$(y_1, \ldots, y_7) = (-1, 0, -1, 1, 1, 0, 0)$$

is a certificate of dual infeasibility (see Section *10.1.2.2*) as we can see from this report:

```
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
```


www.EngineeringBooksPdf.com

Let y^* denote the reported primal solution. **MOSEK** states

- that the problem is *dual infeasible*,
- that the reported solution is a certificate of dual infeasibility, and
- that the infeasibility measure for y^* is approximately zero.

Since the original objective was maximization, we have that $c^T y^* > 0$. See Section *10.1.2* for how to interpret the parameter values in the infeasibility report for a linear program. We see that the variables `y1`, `y3`, `y4`, `y5` and the constraints `x33` and `x34` contribute to infeasibility with non-zero values in the `Activity` column.

One possible strategy to *fix* the infeasibility is to modify the problem so that the certificate of infeasibility becomes invalid. In this case we could do one the following things:

- Add a lower bound on `y3`. This will directly invalidate the certificate of dual infeasibility.
- Increase the object coefficient of `y3`. Changing the coefficients sufficiently will invalidate the inequality $c^T y^* > 0$ and thus the certificate.
- Add lower bounds on `x11` or `x31`. This will directly invalidate the certificate of infeasibility.

Please note that modifying the problem to invalidate the reported certificate does *not* imply that the problem becomes dual feasible — the 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.

14.5 Theory Concerning Infeasible Problems

This section discusses the theory of infeasibility certificates and how **MOSEK** uses a certificate to produce an infeasibility report. In general, **MOSEK** solves the problem

$$\begin{array}{lccccc} \text{minimize} & & & c^T x + c^f & & \\ \text{subject to} & l^c & \leq & Ax & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x \end{array} \tag{14.2}$$

where the corresponding dual problem is

$$\begin{array}{lcc} \text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c & \\ & +(l^x)^T s_l^x - (u^x)^T s_u^x + c^f & \\ \text{subject to} & A^T y + s_l^x - s_u^x = c, & \\ & -y + s_l^c - s_u^c = 0, & \\ & s_l^c, s_u^c, s_l^x, s_u^x \leq 0. & \end{array} \tag{14.3}$$

We use the convension that for any bound that is not finite, the corresponding dual variable is fixed at zero (and thus will have no influence on the dual problem). For example

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

14.6 The Certificate of Primal Infeasibility

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

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

www.EngineeringBooksPdf.com

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

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

and

$$\begin{array}{rcl} A^T y + s_l^{x*} - s_u^{x*} & = & 0, \\ -y + s_l^{c*} - s_u^{c*} & = & 0, \\ s_l^{c*}, s_u^{c*}, s_l^{x*}, s_u^{x*} \leq 0. & & \end{array}$$

The well-known *Farkas Lemma* tells us that (14.2) is infeasible if and only if a certificate of primal infeasibility exists.

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

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

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

$$(s_l^{x*})_j > 0((s_u^{x*})_i > 0)$$

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

14.7 The certificate of dual infeasibility

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

$$\begin{array}{lccccc} \text{minimize} & & & c^T x & & \\ \text{subject to} & \bar{l}^c & \leq & Ax & \leq & \bar{u}^c, \\ & \bar{l}^x & \leq & x & \leq & \bar{u}^x \end{array}$$

with negative objective value, where we use the definitions

$$\bar{l}_i^c := \left\{ \begin{array}{ll} 0, & l_i^c > -\infty, \\ -\infty, & \text{otherwise}, \end{array} \right\}, \; \bar{u}_i^c := \left\{ \begin{array}{ll} 0, & u_i^c < \infty, \\ \infty, & \text{otherwise}, \end{array} \right\}$$

and

$$\bar{l}_i^x := \left\{ \begin{array}{ll} 0, & l_i^x > -\infty, \\ -\infty, & \text{otherwise}, \end{array} \right\} \quad \text{and} \quad \bar{u}_i^x := \left\{ \begin{array}{ll} 0, & u_i^x < \infty, \\ \infty, & \text{otherwise}. \end{array} \right\}$$

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

$$\begin{array}{rcccl} & & c^T x^* & < & 0, \\ \bar{l}^c & \leq & Ax^* & \leq & \bar{u}^c, \\ \bar{l}^x & \leq & x^* & \leq & \bar{u}^x \end{array} \tag{14.4}$$

The well-known Farkas Lemma tells us that (14.3) is infeasible if and only if a certificate of dual infeasibility exists.

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

$$x_j^* \leq 0,$$

variable j is involved in the dual infeasibility.

www.EngineeringBooksPdf.com

CHAPTER

FIFTEEN

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.

15.1 Sensitivity Analysis for Linear Problems

15.1.1 The Optimal Objective Value Function

Assume that we are given the problem

$$\begin{array}{rcllcll} z(l^c,u^c,l^x,u^x,c) & = & \text{minimize} & & & c^T x & \\ & & \text{subject to} & l^c & \leq & Ax & \leq u^c, \\ & & & l^x & \leq & x & \leq u^x, \end{array} \tag{15.1}$$

and we want to know how the optimal objective value changes as l_i^c is perturbed. To answer this question we define the perturbed problem for l_i^c as follows

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

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

$$f_{l_i^c}(\beta) \tag{15.2}$$

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 (15.2) shows the optimal objective value as a function of varying l_i^c with the other bounds fixed.

www.EngineeringBooksPdf.com

It is possible to prove that the function (15.2) is a piecewise linear and convex function, i.e. its graph may look like in Fig. 15.1 and Fig. 15.2.

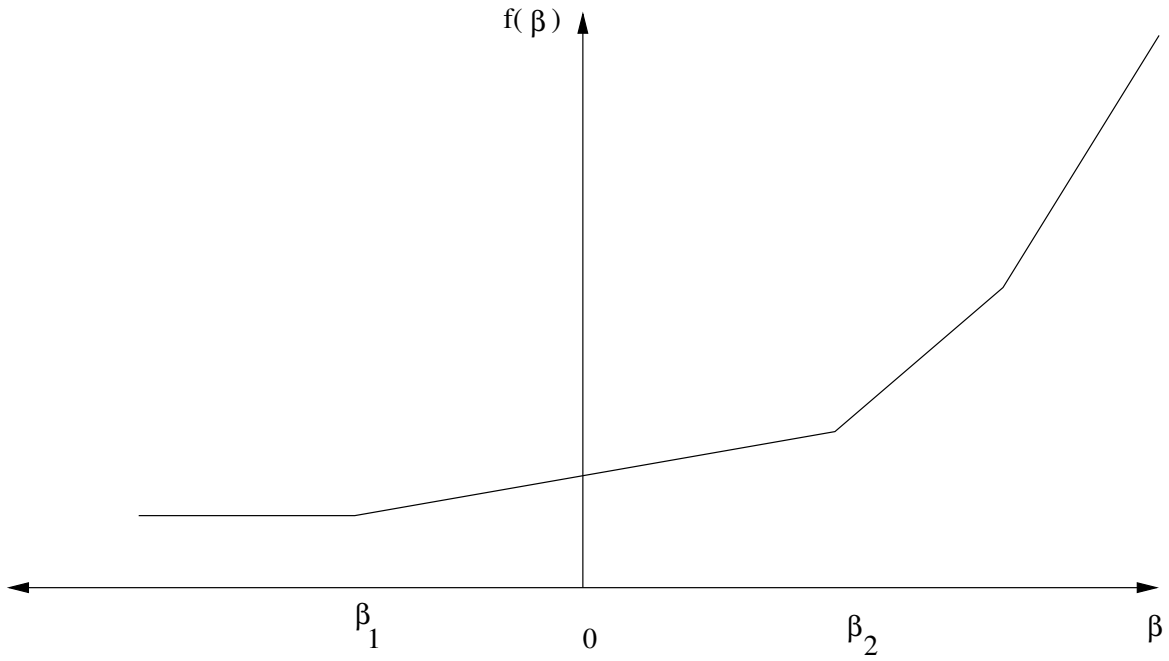


Fig. 15.1: $\beta = 0$ is in the interior of linearity interval.

Clearly, if the function $f_{l_i^c}(\beta)$ does not change much when β is changed, then we can conclude that the optimal objective value is insensitive to changes in l_i^c. Therefore, we are interested in the rate of change in $f_{l_i^c}(\beta)$ for small changes in β — specifically the gradient

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

which is called the *shadow price* related to l_i^c. The shadow price specifies how the objective value changes for small changes of β around zero. Moreover, we are interested in the *linearity interval*

$$\beta \in [\beta_1, \beta_2]$$

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. 15.2. 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 (15.1) 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.

www.EngineeringBooksPdf.com

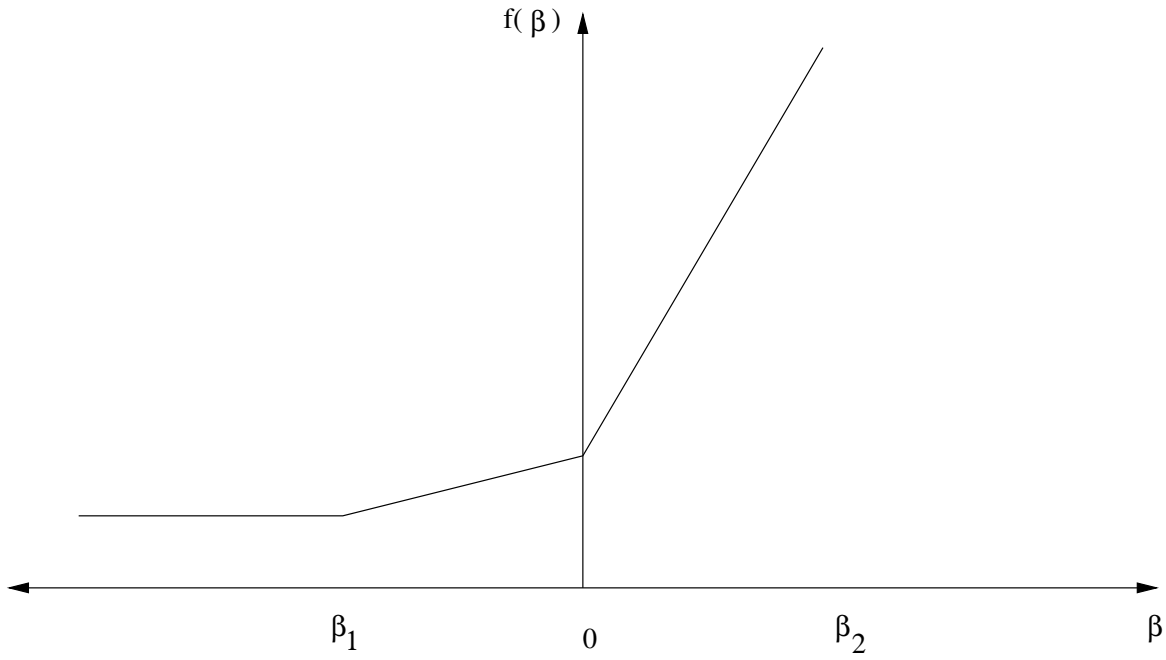


Fig. 15.2: $\beta = 0$ is a breakpoint.

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.

15.1.2 The Basis Type Sensitivity Analysis

The classical sensitivity analysis discussed in most textbooks about linear optimization, e.g. *[Chv83]*, is based on an optimal basic solution or, equivalently, on an optimal basis. This method may produce misleading results *[RTV97]* but is **computationally cheap**. Therefore, and for historical reasons, this method is available in **MOSEK**.

We will now briefly discuss the basis type sensitivity analysis. Given an optimal basic solution which provides a partition of variables into basic and non-basic variables, the basis type sensitivity analysis computes the linearity interval $[\beta_1, \beta_2]$ so that the basis remains optimal for the perturbed problem. A shadow price associated with the linearity interval is also computed. However, it is well-known that an optimal basic solution may not be unique and therefore the result depends on the optimal basic solution employed in the sensitivity analysis. This implies that the computed interval is only a subset of the largest interval for which the shadow price is constant. Furthermore, the optimal objective value function might have a breakpoint for $\beta = 0$. In this case the basis type sensitivity method will only provide a subset of either the left or the right linearity interval.

In summary, the basis type sensitivity analysis is computationally cheap but does not provide complete information. Hence, the results of the basis type sensitivity analysis should be used with care.

www.EngineeringBooksPdf.com

15.1.3 The Optimal Partition Type Sensitivity Analysis

Another method for computing the complete linearity interval is called the *optimal partition type sensitivity analysis*. The main drawback of the optimal partition type sensitivity analysis is that it is computationally expensive compared to the basis type analysis. This type of sensitivity analysis is currently provided as an experimental feature in **MOSEK**.

Given the optimal primal and dual solutions to (15.1), i.e. x^* and $((s_l^c)^*, (s_u^c)^*, (s_l^x)^*, (s_u^x)^*)$ the optimal objective value is given by

$$z^* := c^T x^*.$$

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

$$\begin{array}{rcll} \sigma_1 & = & \text{minimize} & e_i^T s_l^c \\ & & \text{subject to} & A^T(s_l^c - s_u^c) + s_l^x - s_u^x = c, \\ & & & (l^c)^T(s_l^c) - (u^c)^T(s_u^c) + (l^x)^T(s_l^x) - (u^x)^T(s_u^x) = z^*, \\ & & & s_l^c, s_u^c, s_l^c, s_u^x \geq 0 \end{array}$$

and

$$\begin{array}{rcll} \sigma_2 & = & \text{maximize} & e_i^T s_l^c \\ & & \text{subject to} & A^T(s_l^c - s_u^c) + s_l^x - s_u^x = c, \\ & & & (l^c)^T(s_l^c) - (u^c)^T(s_u^c) + (l^x)^T(s_l^x) - (u^x)^T(s_u^x) = z^*, \\ & & & s_l^c, s_u^c, s_l^c, s_u^x \geq 0. \end{array}$$

These two optimization problems make it easy to interpret the shadow price. Indeed, if $((s_l^c)^*, (s_u^c)^*, (s_l^x)^*, (s_u^x)^*)$ is an arbitrary optimal solution then

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

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

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

and

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

The linearity intervals and shadow prices for u_i^c, l_j^x, and u_j^x are computed similarly to l_i^c.

The left and right shadow prices for c_j denoted σ_1 and σ_2 respectively are computed as follows:

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

and

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

Once again the above two optimization problems make it easy to interpret the shadow prices. Indeed, if x^* is an arbitrary primal optimal solution, then

$$x_j^* \in [\sigma_1, \sigma_2].$$

www.EngineeringBooksPdf.com

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

$$\begin{array}{rcll} \beta_1 & = & \text{minimize} & \beta \\ & & \text{subject to} & A^T(s_l^c - s_u^c) + s_l^x - s_u^x = c + \beta e_j, \\ & & & (l^c)^T(s_l^c) - (u^c)^T(s_u^c) + (l^x)^T(s_l^x) - (u^x)^T(s_u^x) - \sigma_1\beta \leq z^*, \\ & & & s_l^c, s_u^c, s_l^c, s_u^x \geq 0 \end{array}$$

and

$$\begin{array}{rcll} \beta_2 & = & \text{maximize} & \beta \\ & & \text{subject to} & A^T(s_l^c - s_u^c) + s_l^x - s_u^x = c + \beta e_j, \\ & & & (l^c)^T(s_l^c) - (u^c)^T(s_u^c) + (l^x)^T(s_l^x) - (u^x)^T(s_u^x) - \sigma_2\beta \leq z^*, \\ & & & s_l^c, s_u^c, s_l^c, s_u^x \geq 0. \end{array}$$

15.1.4 Example: Sensitivity Analysis

As an example we will use the following transportation problem. Consider the problem of minimizing the transportation cost between a number of production plants and stores. Each plant supplies a number of goods and each store has a given demand that must be met. Supply, demand and cost of transportation per unit are shown in Fig. 15.3.

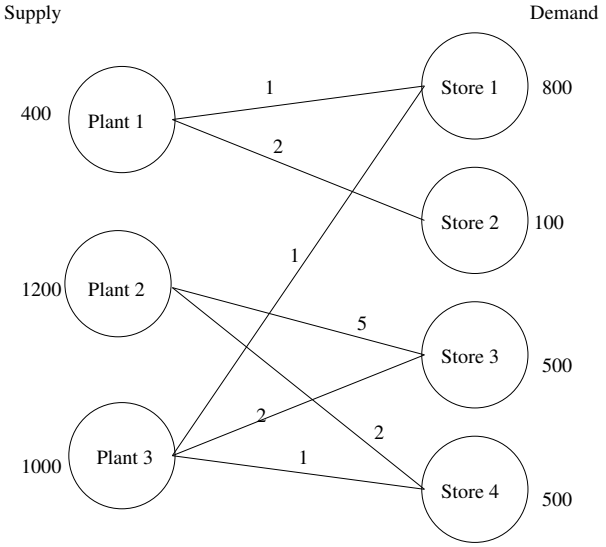


Fig. 15.3: Supply, demand and cost of transportation.

If we denote the number of transported goods from location i to location j by x_{ij}, problem can be formulated as the linear optimization problem of minimizing

$$1x_{11} + 2x_{12} + 5x_{23} + 2x_{24} + 1x_{31} + 2x_{33} + 1x_{34}$$

subject to

$$\begin{array}{rrrrrrrrrrrrr} x_{11} & + & x_{12} & & & & & & & & & \leq & 400, \\ & & & x_{23} & + & x_{24} & & & & & & \leq & 1200, \\ & & & & & & x_{31} & + & x_{33} & + & x_{34} & \leq & 1000, \\ x_{11} & & & & & & + & x_{31} & & & & = & 800, \\ & & x_{12} & & & & & & & & & = & 100, \\ & & & x_{23} & + & & & & x_{33} & & & = & 500, \\ & & & & & x_{24} & + & & & & x_{34} & = & 500, \\ x_{11}, & & x_{12}, & & x_{23}, & & x_{24}, & & x_{31}, & & x_{33}, & & x_{34} \geq 0. \end{array} \tag{15.3}$$

The sensitivity parameters are shown in Table 15.1 and Table 15.2 for the basis type analysis and in Table 15.3 and Table 15.4 for the optimal partition type analysis.

www.EngineeringBooksPdf.com

Table 15.1: Ranges and shadow prices related to bounds on constraints and variables: results for the basis type sensitivity analysis.

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 15.2: Ranges and shadow prices related to bounds on constraints and variables: results for the optimal partition type sensitivity analysis.

Con.	β_1	β_2	σ_1	σ_2
1	-300.00	500.00	3.00	1.00
2	-700.00	$+\infty$	-0.00	-0.00
3	-500.00	500.00	3.00	1.00
4	-500.00	500.00	2.00	4.00
5	-100.00	300.00	3.00	5.00
6	-500.00	700.00	3.00	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}	$-\infty$	500.00	0.00	2.00
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}	$-\infty$	500.00	0.00	2.00

www.EngineeringBooksPdf.com

Table 15.3: Ranges and shadow prices related to the objective coefficients: results for the basis type sensitivity analysis.

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

Table 15.4: Ranges and shadow prices related to the objective coefficients: results for the optimal partition type sensitivity analysis.

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 optimal partition type sensitivity analysis we see that for constraint number 1 we have $\sigma_1 = 3$, $\sigma_2 = 1$ and $\beta_1 = -300$, $\beta_2 = 500$. Therefore, we have a left linearity interval of $[-300, 0]$ and a right interval of $[0, 500]$. The corresponding left and right shadow prices are 3 and 1 respectively. This implies that if the upper bound on constraint 1 increases by

$$\beta \in [0, \beta_1] = [0, 500]$$

then the optimal objective value will decrease by the value

$$\sigma_2 \beta = 1\beta.$$

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

$$\beta \in [0, 300]$$

then the optimal objective value will increase by the value

$$\sigma_1 \beta = 3\beta.$$

15.2 Sensitivity Analysis with MOSEK

MOSEK provides the functions *task.primalsensitivity* and *task.dualsensitivity* for performing sensitivity analysis. The code in Listing 15.1 gives an example of its use.

www.EngineeringBooksPdf.com

Listing 15.1: 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()



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


www.EngineeringBooksPdf.com

```
                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,
                            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 ]
```


www.EngineeringBooksPdf.com

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

        print ('Results from sensitivity analysis on objective coefficients:')

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

    return None

# call the main function
try:
    main ()
except mosek.Exception 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)
```


www.EngineeringBooksPdf.com

CHAPTER

SIXTEEN

API REFERENCE

This section contains the complete reference of the **MOSEK** Optimizer API for Python. It is organized as follows:

- *General API conventions.*
- API functionalities either *grouped by topic* or accessable via the complete class list:
  - *Env*
  - *Task*
- *Exceptions*
- *SCopt interface*
- *Optimizer parameters*
- *Response codes*
- *Constants*

16.1 API Conventions

16.1.1 Naming Conventions for Arguments

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 16.1 the variable names used to specify the problem parameters are listed.

www.EngineeringBooksPdf.com

Table 16.1: Naming convensions 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
numqonz	int		q^o_{ij}
qosubi	int[]	numqonz	q^o_{ij}
qosubj	int[]	numqonz	q^o_{ij}
qoval	float[]	numqonz	q^o_{ij}
c	float[]	numvar	c_j
cfix	float		c^f
numqcnz	int		q^k_{ij}
qcsubk	int[]	qcnz	q^k_{ij}
qcsubi	int[]	qcnz	q^k_{ij}
qcsubj	int[]	qcnz	q^k_{ij}
qcval	float[]	qcnz	q^k_{ij}
aptrb	int[]	numvar	a_{ij}
aptre	int[]	numvar	a_{ij}
asub	int[]	aptre[numvar-1]	a_{ij}
aval	float[]	aptre[numvar-1]	a_{ij}
bkc	int[]	numcon	l^c_k and u^c_k
blc	float[]	numcon	l^c_k
buc	float[]	numcon	u^c_k
bkx	int[]	numvar	l^x_k and u^x_k
blx	float[]	numvar	l^x_k
bux	float[]	numvar	u^x_k

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

- The quadratic terms in the objective: $q^o_{\texttt{qosubi[t]},\texttt{qosubj[t]}} = \texttt{qoval[t]}, \quad t = 0, \ldots, \texttt{numqonz} - 1$.
- The linear terms in the objective : $c_j = \texttt{c[j]}, \quad j = 0, \ldots, \texttt{numvar} - 1$
- The fixed term in the objective : $c^f = \texttt{cfix}$.
- The quadratic terms in the constraints: $q^{\texttt{qcsubk[t]}}_{\texttt{qcsubi[t]},\texttt{qcsubj[t]}} = \texttt{qcval[t]}, \quad t = 0, \ldots, \texttt{numqcnz} - 1$
- The linear terms in the constraints: $a_{\texttt{asub[t]},\texttt{j}} = \texttt{aval[t]}, \quad t = \texttt{ptrb[j]}, \ldots, \texttt{ptre[j]} - 1, j = 0, \ldots, \texttt{numvar} - 1$

The bounds on the constraints are specified using the variables `bkc`, `blc`, and `buc`. The components of the integer array `bkc` specify the bound type according to Table 16.2

Table 16.2: 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 $-\infty < l^c_2$ and $u^c_2 = \infty$. Finally, the numerical values of the bounds are given by

$$l^c_k = \texttt{blc[k]}, \quad k = 0, \ldots, \texttt{numcon} - 1$$

www.EngineeringBooksPdf.com

and

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

The bounds on the variables are specified using the variables `bkx`, `blx`, and `bux`. The components in the integer array `bkx` specify the bound type according to Table Table 16.2. The numerical values for the lower bounds on the variables are given by

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

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

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

Bounds

A bound on a variable or on a constraint in **MOSEK** consists of a *bound key*, as defined in Table 16.2, a lower bound value and an upper bound value. Even if a variable or constraint is bounded only from below, e.g. $x \geq 0$, both bounds are inputted or extracted; the value inputted as upper bound for $(x \geq 0)$ is ignored.

16.1.2 Vector Formats

Three different vector formats are used in the **MOSEK** API:

Full vector

This is simply an array where the first element corresponds to the first item, the second element to the second item etc. For example to get the linear coefficients of the objective in `task`, one would write

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

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

Vector slice

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

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

task.getboundslice(accmode.con, 2, 10,
                   bound_key,lower_bound,upper_bound)
```

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

Sparse vector

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

www.EngineeringBooksPdf.com

```
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.putboundlist(accmode.con, bound_index,
                  bound_key,lower_bound,upper_bound)
```

Note that the list of indexes need not be ordered.

16.1.3 Matrix Formats

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

Unordered Triplets

In unordered triplet format each entry is defined as a row index, a column index and a coefficient. For example, to input the A matrix coefficients for $a_{1,2} = 1.1, a_{3,3} = 4.3$, and $a_{5,4} = 0.2$, one would write as follows:

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

Please note that in some cases (like *task.putaijlist*) *only* the specified indexes remain modified — all other are unchanged. In other cases (such as *task.putqconk*) the triplet format is used to modify *all* entries — entries that are not specified are set to 0.

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. In the column-wise format the position of the non-zeros are given as a list of row indexes. In the row-wise format the position of the non-zeros are given as a list of column indexes. Values of the non-zero entries are given in column or row order.

Column ordered sparse matrix

A sparse matrix in column ordered format consists of:

- `asub`: List of row indexes.
- `aval`: List of non-zero entries of A ordered by columns.
- `ptrb`: Where `ptrb[j]` is the position of the first value/index in `aval` / `asub` for column j.
- `ptre`: Where `ptre[j]` is the position of the last value/index plus one in `aval` / `asub` for column j.

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

$$j = 0, \ldots, numcol - 1.$$

We define

$$a_{\texttt{asub}[k],j} = \texttt{aval}[k], k = \texttt{ptrb}[j], \ldots, \texttt{ptre}[j] - 1.$$

www.EngineeringBooksPdf.com

As an example consider the matrix

$$A = \begin{bmatrix} 1.1 & & 1.3 & 1.4 & \\ & 2.2 & & & 2.5 \\ 3.1 & & & 3.4 & \\ & & 4.4 & & \end{bmatrix} \tag{16.1}$$

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

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

Fig. 16.1 illustrates how the matrix A in (16.1) is represented in column ordered sparse matrix format.

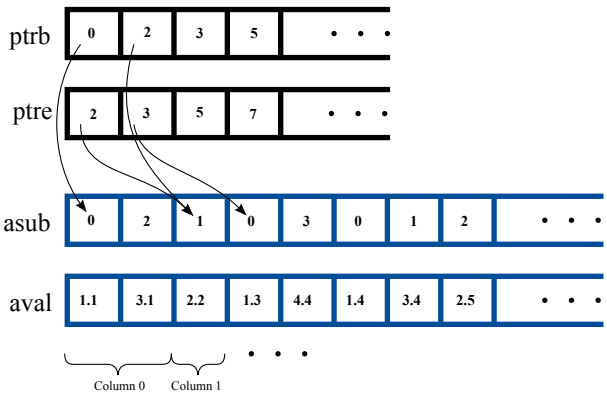


Fig. 16.1: The matrix A (16.1) represented in column ordered packed sparse matrix format.

Row ordered sparse matrix

The matrix A (16.1) can also be represented in the row ordered sparse matrix format as:

$$\begin{array}{rcl} \texttt{ptrb} & = & [0, 3, 5, 7], \\ \texttt{ptre} & = & [3, 5, 7, 8], \\ \texttt{asub} & = & [0, 2, 3, 1, 4, 0, 3, 2], \\ \texttt{aval} & = & [1.1, 1.3, 1.4, 2.2, 2.5, 3.1, 3.4, 4.4]. \end{array}$$

16.2 Functions grouped by topic

16.2.1 Bounds

- *task.getconbound*
- *task.getvarbound*
- *task.getbound*
- *task.getconboundslice*
- *task.getvarboundslice*
- *task.getboundslice*
- *task.putboundslice*
- *task.putbound*

www.EngineeringBooksPdf.com

- *task.putboundlist*
- *task.putconboundlist*
- *task.putvarboundlist*

16.2.2 Diagnosing infeasibility

- *task.getinfeasiblesubproblem*

16.2.3 Inputting solution values

- *task.putskc*
- *task.putskx*
- *task.putxc*
- *task.putxx*
- *task.puty*
- *task.putslc*
- *task.putsuc*
- *task.putslx*
- *task.putsux*
- *task.putsnx*
- *task.putskcslice*
- *task.putskxslice*
- *task.putxcslice*
- *task.putxxslice*
- *task.putyslice*
- *task.putslcslice*
- *task.putsucslice*
- *task.putslxslice*
- *task.putsuxslice*
- *task.putsnxslice*
- *task.putsolution*
- *task.putsolutioni*

16.2.4 Linear algebra utility functions for performing linear algebra operations

- *env.computesparsecholesky*
- *env.sparsetriangularsolvedense*

www.EngineeringBooksPdf.com

16.2.5 Management of parameters

- *task.getnumparam*
- *task.isdouparname*
- *task.isintparname*
- *task.isstrparname*
- *task.setdefaults*

16.2.6 Management of the environment

- *env.checkoutlicense*
- *env.checkinlicense*
- *env.putlicensedebug*
- *env.putlicensecode*
- *env.putlicensewait*
- *env.putlicensepath*

16.2.7 Memory allocation and deallocation

- *task.checkmem*
- *task.getmemusage*

16.2.8 Naming

- *task.getbarvarnamelen*
- *task.getbarvarname*
- *task.getbarvarnameindex*
- *task.putconname*
- *task.putvarname*
- *task.putconename*
- *task.getvarnamelen*
- *task.getvarname*
- *task.getconnamelen*
- *task.getconname*
- *task.getconnameindex*
- *task.getvarnameindex*
- *task.getconenamelen*
- *task.getconename*
- *task.getconenameindex*
- *task.getobjname*
- *task.gettasknamelen*
- *task.gettaskname*

www.EngineeringBooksPdf.com

- *task.putobjname*
- *task.puttaskname*

16.2.9 Obtain information about the solutions.

- *task.getdualobj*
- *task.getsolsta*
- *task.getprosta*
- *task.getsolutioninfo*
- *task.solutiondef*

16.2.10 Obtaining solution values

- *task.getsolution*
- *task.getsolutioni*
- *task.getskc*
- *task.getskx*
- *task.getxc*
- *task.getxx*
- *task.gety*
- *task.getslc*
- *task.getsuc*
- *task.getslx*
- *task.getsux*
- *task.getsnx*
- *task.getskcslice*
- *task.getskxslice*
- *task.getxcslice*
- *task.getxxslice*
- *task.getyslice*
- *task.getslcslice*
- *task.getsucslice*
- *task.getslxslice*
- *task.getsuxslice*
- *task.getsnxslice*
- *task.getsolutionslice*
- *task.getreducedcosts*

www.EngineeringBooksPdf.com

16.2.11 Obtaining task parameters values

- *task.getdouparam*
- *task.getintparam*
- *task.getstrparam*
- *task.getstrparamlen*

16.2.12 Operate on data associated with objective.

- *task.putcfix*
- *task.putobjsense*
- *task.getobjsense*

16.2.13 Operate on data associated with scalar variables

- *task.appendvars*
- *task.removevars*
- *task.getaij*
- *task.getacolnumnz*
- *task.getacol*
- *task.getarownumnz*
- *task.getarow*
- *task.getaslice*
- *task.getarowslicetrip*
- *task.getacolslicetrip*
- *task.getc*
- *task.getcfix*
- *task.getcslice*
- *task.getmaxnumanz*
- *task.getmaxnumvar*
- *task.getnumanz*
- *task.getnumanz64*
- *task.getnumintvar*
- *task.getnumqconknz*
- *task.getnumqobjnz*
- *task.getnumvar*
- *task.getlenbarvarj*
- *task.getqconk*
- *task.getqobj*
- *task.getqobjij*
- *task.getvartype*

www.EngineeringBooksPdf.com

- *task.getvartypelist*
- *task.commitchanges*
- *task.putaij*
- *task.putacol*
- *task.putarow*
- *task.putarowslice*
- *task.putarowlist*
- *task.putacolslice*
- *task.putacollist*
- *task.putvarbound*
- *task.putvarboundslice*
- *task.putclist*
- *task.putcslice*
- *task.getsymmatinfo*
- *task.getnumsymmat*
- *task.getsparsesymmat*
- *task.putmaxnumvar*
- *task.putmaxnumanz*
- *task.putmaxnumqnz*
- *task.getmaxnumqnz*
- *task.putqcon*
- *task.putqconk*
- *task.putqobj*
- *task.putqobjij*
- *task.putvartype*
- *task.putvartypelist*

16.2.14 Operate on data associated with symmetric matrix variables

- *task.removebarvars*
- *task.appendbarvars*
- *task.getmaxnumbarvar*
- *task.getdimbarvarj*
- *task.getbarcidxj*
- *task.appendsparsesymmat*
- *task.putmaxnumbarvar*

www.EngineeringBooksPdf.com

16.2.15 Operate on data associated with the conic constraints

- *task.removecones*
- *task.appendcone*
- *task.appendconeseq*
- *task.appendconesseq*
- *task.getcone*
- *task.getconeinfo*
- *task.getnumcone*
- *task.getnumconemem*
- *task.putcone*

16.2.16 Operate on data associated with the constraints

- *task.appendcons*
- *task.removecons*
- *task.getmaxnumcon*
- *task.getnumcon*
- *task.putconbound*
- *task.putconboundslice*

16.2.17 Operation associated with the basis matrix

- *task.initbasissolve*
- *task.solvewithbasis*
- *task.basiscond*

16.2.18 Optimization

- *task.optimize*

16.2.19 Optimizer statistics

- *task.getdouinf*
- *task.getintinf*
- *task.getlintinf*

16.2.20 Output stream functions

- *task.linkfiletostream*
- *env.linkfiletostream*

www.EngineeringBooksPdf.com

16.2.21 Reading and writing data files

- *task.readdataformat*
- *task.readdata*
- *task.readparamfile*
- *task.readsolution*
- *task.writedata*
- *task.writeparamfile*
- *task.writesolution*
- *task.writejsonsol*

16.2.22 Sensitivity analysis

- *task.primalsensitivity*
- *task.sensitivityreport*
- *task.dualsensitivity*

16.2.23 Setting task parameter values

- *task.putdouparam*
- *task.putintparam*
- *task.putnadouparam*
- *task.putnaintparam*
- *task.putnastrparam*
- *task.putparam*
- *task.putstrparam*

16.2.24 Task diagnostics

- *task.analyzeproblem*
- *task.getprobtype*
- *task.printdata*
- *task.readsummary*
- *task.optimizersummary*
- *task.checkconvexity*
- *env.echointro*

16.2.25 Task management.

- *task.inputdata*
- *task.putmaxnumcon*
- *task.putmaxnumcone*
- *task.getmaxnumcone*

www.EngineeringBooksPdf.com

- *task.deletesolution*
- *env.getcodedesc*

16.3 The Interface for Separable Convex Optimization

SCopt is an easy-to-use interface to the nonlinear optimizer when solving separable convex problems. As currently implemented, SCopt is not capable of handling arbitrary nonlinear expressions. In fact SCopt can handle only the nonlinear expressions $x\log(x)$, e^x, $\log(x)$, and x^g. However, it should be fairly easy to extend the interface to other nonlinear function of a single variable if needed.

16.3.1 Design Principles of SCopt

All the linear data of the problem, such as c and A, is inputted to **MOSEK** as usual, i.e. using the relevant functions in the **MOSEK** API.

The nonlinear part of the problem is specified using some arrays which indicate the type of the nonlinear expressions and where these should be added.

For example given the three `int` arrays — `oprc`, `opric`, and `oprjc` — and the three `double` arrays — `oprfc`, `oprgc` and `oprhc` — the nonlinear expressions in the constraints can be coded in those arrays using the following table:

`oprc[k]`	`opric[k]`	`oprjc[k]`	`oprfc[k]`	`oprgc[k]`	`oprhc[k]`	Expression added to constraint i
0	i	j	f	g	h	$fx_j\ln(x_j)$
1	i	j	f	g	h	fe^{gx_j+h}
2	i	j	f	g	h	$f\ln(gx_j+h)$
3	i	j	f	g	h	$f(x_j+h)^g$

Hence, `oprc[k]` specifies the nonlinear expression type, `opric[k]` indicates to which constraint the nonlinear expression should be added. `oprfc[k]`, `oprgc[k]` and `oprhc[k]` are parameters used when the nonlinear expression is evaluated. This implies that nonlinear expressions can be added to an arbitrary constraint and hence you can create multiple nonlinear constraints.

Using the same method all the nonlinear terms in the objective can be specified using `opro[k]`, `oprjo[k]`, `oprfo[k]`, `oprgo[k]` and `oprho[k]` as shown below:

`opro[k]`	`oprjo[k]`	`oprfo[k]`	`oprgo[k]`	`oprho[k]`	Expression added in objective
0	j	f	g	h	$fx_j\ln(x_j)$
1	j	f	g	h	fe^{gx_j+h}
2	j	f	g	h	$f\ln(gx_j+h)$
3	j	f	g	h	$f(x_j+h)^g$

16.4 Parameters

All parameters (alphabetical order)

- *double parameters*
- *integer parameters*
- *string parameters*

Parameters grouped by topic

Note: some parameters may appear in more than one group.

www.EngineeringBooksPdf.com

- *Overall solver*
- *Mixed-integer optimization*
- *Output information*
- *Optimization system*
- *Simplex optimizer*
- *Logging*
- *Data check*
- *Progress call-back*
- *Analysis*
- *Conic interior-point method*
- *Nonlinear convex method*
- *Solution input/output*
- *Interior-point method*
- *Dual simplex optimizer*
- *Termination criterion*
- *Primal simplex optimizer*
- *Basis identification*
- *License manager*
- *Presolve*
- *Debugging*
- *Data input/output*
- *Infeasibility report*

16.4.1 Parameters List (alphabetically)

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.

`Accepted Values:` [0.0 ;+inf]

`Default Value:` 1e-6

`Groups:` *Analysis*

`dparam.basis_rel_tol_s`
Maximum relative dual bound violation allowed in an optimal basic solution.

`Accepted Values:` [0.0 ;+inf]

`Default Value:` 1.0e-12

`Groups:` *Simplex optimizer*, *Termination criterion*

`dparam.basis_tol_s`
Maximum absolute dual bound violation in an optimal basic solution.

`Accepted Values:` [1.0e-9 ;+inf]

`Default Value:` 1.0e-6

www.EngineeringBooksPdf.com

Groups: *Simplex optimizer*, *Termination criterion*

`dparam.basis_tol_x`
Maximum absolute primal bound violation allowed in an optimal basic solution.

Accepted Values: [1.0e-9 ;+inf]

Default Value: 1.0e-6

Groups: *Simplex optimizer*, *Termination criterion*

`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:

$$d_i \leq -|Q_{ii}|\texttt{check_convexity_rel_tol}$$

Accepted Values: [0 ;+inf]

Default Value: 1e-10

Groups: *Interior-point method*

`dparam.data_sym_mat_tol`
Absolute zero tolerance for elements in in suymmetric matrixes. 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.

Accepted Values: [1.0e-16 ;1.0e-6]

Default Value: 1.0e-12

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.

Accepted Values: [0.0 ;+inf]

Default Value: 1.0e20

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.

Accepted Values: [0.0 ;+inf]

Default Value: 1.0e10

Groups: *Data check*

`dparam.data_tol_aij`
Absolute zero tolerance for elements in A. If any value A_{ij} is smaller than this parameter in absolute terms **MOSEK** will treat the values as zero and generate a warning.

Accepted Values: [1.0e-16 ;1.0e-6]

Default Value: 1.0e-12

Groups: *Data check*

www.EngineeringBooksPdf.com

`dparam.data_tol_aij_huge`
An element in A which is larger than this value in absolute size causes an error.

`Accepted Values:` [0.0 ;+inf]

`Default Value:` 1.0e20

`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.

`Accepted Values:` [0.0 ;+inf]

`Default Value:` 1.0e10

`Groups:` *Data check*

`dparam.data_tol_bound_inf`
Any bound which in absolute value is greater than this parameter is considered infinite.

`Accepted Values:` [0.0 ;+inf]

`Default Value:` 1.0e16

`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.

`Accepted Values:` [0.0 ;+inf]

`Default Value:` 1.0e8

`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.

`Accepted Values:` [0.0 ;+inf]

`Default Value:` 1.0e8

`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.

`Accepted Values:` [0.0 ;+inf]

`Default Value:` 1.0e16

`Groups:` *Data check*

`dparam.data_tol_qij`
Absolute zero tolerance for elements in Q matrices.

`Accepted Values:` [0.0 ;+inf]

`Default Value:` 1.0e-16

`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.

`Accepted Values:` [0.0 ;+inf]

`Default Value:` 1.0e-8

www.EngineeringBooksPdf.com

Groups: *Data check*

`dparam.intpnt_co_tol_dfeas`
Dual feasibility tolerance used by the conic interior-point optimizer.

Accepted Values: [0.0 ;1.0]

Default Value: 1.0e-8

Groups: *Interior-point method*, *Termination criterion*, *Conic interior-point method*

`dparam.intpnt_co_tol_infeas`
Controls when the conic interior-point optimizer declares the model primal or dual infeasible. A small number means the optimizer gets more conservative about declaring the model infeasible.

Accepted Values: [0.0 ;1.0]

Default Value: 1.0e-10

Groups: *Interior-point method*, *Termination criterion*, *Conic interior-point method*

`dparam.intpnt_co_tol_mu_red`
Relative complementarity gap feasibility tolerance used by the conic interior-point optimizer.

Accepted Values: [0.0 ;1.0]

Default Value: 1.0e-8

Groups: *Interior-point method*, *Termination criterion*, *Conic interior-point method*

`dparam.intpnt_co_tol_near_rel`
If **MOSEK** cannot compute a solution that has the prescribed accuracy, then it will multiply the termination tolerances with value of this parameter. If the solution then satisfies the termination criteria, then the solution is denoted near optimal, near feasible and so forth.

Accepted Values: [1.0 ;+inf]

Default Value: 1000

Groups: *Interior-point method*, *Termination criterion*, *Conic interior-point method*

`dparam.intpnt_co_tol_pfeas`
Primal feasibility tolerance used by the conic interior-point optimizer.

Accepted Values: [0.0 ;1.0]

Default Value: 1.0e-8

Groups: *Interior-point method*, *Termination criterion*, *Conic interior-point method*

`dparam.intpnt_co_tol_rel_gap`
Relative gap termination tolerance used by the conic interior-point optimizer.

Accepted Values: [0.0 ;1.0]

Default Value: 1.0e-7

Groups: *Interior-point method*, *Termination criterion*, *Conic interior-point method*

`dparam.intpnt_nl_merit_bal`
Controls if the complementarity and infeasibility is converging to zero at about equal rates.

Accepted Values: [0.0 ;0.99]

Default Value: 1.0e-4

Groups: *Interior-point method*, *Nonlinear convex method*

`dparam.intpnt_nl_tol_dfeas`
Dual feasibility tolerance used when a nonlinear model is solved.

Accepted Values: [0.0 ;1.0]

Default Value: 1.0e-8

www.EngineeringBooksPdf.com

Groups: *Interior-point method*, *Termination criterion*, *Nonlinear convex method*

`dparam.intpnt_nl_tol_mu_red`
Relative complementarity gap tolerance.

Accepted Values: [0.0 ;1.0]

Default Value: 1.0e-12

Groups: *Interior-point method*, *Termination criterion*, *Nonlinear convex method*

`dparam.intpnt_nl_tol_near_rel`
If the **MOSEK** nonlinear interior-point optimizer cannot compute a solution that has the prescribed accuracy, then it will multiply the termination tolerances with value of this parameter. If the solution then satisfies the termination criteria, then the solution is denoted near optimal, near feasible and so forth.

Accepted Values: [1.0 ;+inf]

Default Value: 1000.0

Groups: *Interior-point method*, *Termination criterion*, *Nonlinear convex method*

`dparam.intpnt_nl_tol_pfeas`
Primal feasibility tolerance used when a nonlinear model is solved.

Accepted Values: [0.0 ;1.0]

Default Value: 1.0e-8

Groups: *Interior-point method*, *Termination criterion*, *Nonlinear convex method*

`dparam.intpnt_nl_tol_rel_gap`
Relative gap termination tolerance for nonlinear problems.

Accepted Values: [1.0e-14 ;+inf]

Default Value: 1.0e-6

Groups: *Termination criterion*, *Interior-point method*, *Nonlinear convex method*

`dparam.intpnt_nl_tol_rel_step`
Relative step size to the boundary for general nonlinear optimization problems.

Accepted Values: [1.0e-4 ;0.9999999]

Default Value: 0.995

Groups: *Interior-point method*, *Nonlinear convex method*

`dparam.intpnt_qo_tol_dfeas`
Dual feasibility tolerance used when the interior-point optimizer is applied to a quadratic optimization problem..

Accepted Values: [0.0 ;1.0]

Default Value: 1.0e-8

Groups: *Interior-point method*, *Termination criterion*

`dparam.intpnt_qo_tol_infeas`
Controls when the conic interior-point optimizer declares the model primal or dual infeasible. A small number means the optimizer gets more conservative about declaring the model infeasible.

Accepted Values: [0.0 ;1.0]

Default Value: 1.0e-10

Groups: *Interior-point method*, *Termination criterion*

www.EngineeringBooksPdf.com

`dparam.intpnt_qo_tol_mu_red`
Relative complementarity gap feasibility tolerance used when interior-point optimizer is applied to a quadratic optimization problem.

`Accepted Values:` [0.0 ;1.0]

`Default Value:` 1.0e-8

`Groups:` *Interior-point method*, *Termination criterion*

`dparam.intpnt_qo_tol_near_rel`
If **MOSEK** cannot compute a solution that has the prescribed accuracy, then it will multiply the termination tolerances with value of this parameter. If the solution then satisfies the termination criteria, then the solution is denoted near optimal, near feasible and so forth.

`Accepted Values:` [1.0 ;+inf]

`Default Value:` 1000

`Groups:` *Interior-point method*, *Termination criterion*

`dparam.intpnt_qo_tol_pfeas`
Primal feasibility tolerance used when the interior-point optimizer is applied to a quadratic optimization problem.

`Accepted Values:` [0.0 ;1.0]

`Default Value:` 1.0e-8

`Groups:` *Interior-point method*, *Termination criterion*

`dparam.intpnt_qo_tol_rel_gap`
Relative gap termination tolerance used when the interior-point optimizer is applied to a quadratic optimization problem.

`Accepted Values:` [0.0 ;1.0]

`Default Value:` 1.0e-8

`Groups:` *Interior-point method*, *Termination criterion*

`dparam.intpnt_tol_dfeas`
Dual feasibility tolerance used for linear and quadratic optimization problems.

`Accepted Values:` [0.0 ;1.0]

`Default Value:` 1.0e-8

`Groups:` *Interior-point method*, *Termination criterion*

`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.

`Accepted Values:` [1.0e-4 ;+inf]

`Default Value:` 1.0

`Groups:` *Interior-point method*

`dparam.intpnt_tol_infeas`
Controls when the optimizer declares the model primal or dual infeasible. A small number means the optimizer gets more conservative about declaring the model infeasible. A value of 0.0 means the optimizer must have an exact certificate of infeasibility and this is very unlikely to happen.

`Accepted Values:` [0.0 ;1.0]

`Default Value:` 1.0e-10

`Groups:` *Interior-point method*, *Termination criterion*, *Nonlinear convex method*

www.EngineeringBooksPdf.com

`dparam.intpnt_tol_mu_red`
Relative complementarity gap tolerance.

`Accepted Values:` [0.0 ;1.0]

`Default Value:` 1.0e-16

`Groups:` *Interior-point method*, *Termination criterion*

`dparam.intpnt_tol_path`
Controls how close the interior-point optimizer follows the central path. A large value of this parameter means the central is followed very closely. On numerical unstable problems it may be worthwhile to increase this parameter.

`Accepted Values:` [0.0 ;0.9999]

`Default Value:` 1.0e-8

`Groups:` *Interior-point method*

`dparam.intpnt_tol_pfeas`
Primal feasibility tolerance used for linear and quadratic optimization problems.

`Accepted Values:` [0.0 ;1.0]

`Default Value:` 1.0e-8

`Groups:` *Interior-point method*, *Termination criterion*

`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.

`Accepted Values:` [1.0e-4 ;+inf]

`Default Value:` 1.0

`Groups:` *Interior-point method*

`dparam.intpnt_tol_rel_gap`
Relative gap termination tolerance.

`Accepted Values:` [1.0e-14 ;+inf]

`Default Value:` 1.0e-8

`Groups:` *Termination criterion*, *Interior-point method*

`dparam.intpnt_tol_rel_step`
Relative step size to the boundary for linear and quadratic optimization problems.

`Accepted Values:` [1.0e-4 ;0.999999]

`Default Value:` 0.9999

`Groups:` *Interior-point method*

`dparam.intpnt_tol_step_size`
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 stop.

`Accepted Values:` [0.0 ;1.0]

`Default Value:` 1.0e-6

`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

www.EngineeringBooksPdf.com

outside, the interval [*dparam.lower_obj_cut* , *dparam.upper_obj_cut*], then **MOSEK** is terminated.

`Accepted Values:` [-inf ;+inf]

`Default Value:` -1.0e30

`Groups:` *Termination criterion*

`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 $-\infty$.

`Accepted Values:` [-inf ;+inf]

`Default Value:` -0.5e30

`Groups:` *Termination criterion*

`dparam.mio_disable_term_time`
This parameter specifies the number of seconds n during which the termination criteria governed by

- *iparam.mio_max_num_relaxs*
- *iparam.mio_max_num_branches*
- *dparam.mio_near_tol_abs_gap*
- *dparam.mio_near_tol_rel_gap*

is disabled since the beginning of the optimization.

A negative value is identical to infinity i.e. the termination criteria are never checked.

`Accepted Values:` [-inf ;+inf]

`Default Value:` -1.0

`Groups:` *Mixed-integer optimization*, *Termination criterion*

`dparam.mio_max_time`
This parameter limits the maximum time spent by the mixed-integer optimizer. A negative number means infinity.

`Accepted Values:` [-inf ;+inf]

`Default Value:` -1.0

`Groups:` *Mixed-integer optimization*, *Termination criterion*

`dparam.mio_near_tol_abs_gap`
Relaxed absolute optimality tolerance employed by the mixed-integer optimizer. This termination criteria is delayed. See *dparam.mio_disable_term_time* for details.

`Accepted Values:` [0.0 ;+inf]

`Default Value:` 0.0

`Groups:` *Mixed-integer optimization*

`dparam.mio_near_tol_rel_gap`
The mixed-integer optimizer is terminated when this tolerance is satisfied. This termination criteria is delayed. See *dparam.mio_disable_term_time* for details.

`Accepted Values:` [0.0 ;+inf]

`Default Value:` 1.0e-3

`Groups:` *Mixed-integer optimization*, *Termination criterion*

www.EngineeringBooksPdf.com

`dparam.mio_rel_gap_const`
This value is used to compute the relative gap for the solution to an integer optimization problem.

`Accepted Values:` [1.0e-15 ;+inf]

`Default Value:` 1.0e-10

`Groups:` *Mixed-integer optimization*, *Termination criterion*

`dparam.mio_tol_abs_gap`
Absolute optimality tolerance employed by the mixed-integer optimizer.

`Accepted Values:` [0.0 ;+inf]

`Default Value:` 0.0

`Groups:` *Mixed-integer optimization*

`dparam.mio_tol_abs_relax_int`
Absolute relaxation tolerance of the integer constraints. I.e. $\min(|x| - \lfloor x \rfloor, \lceil x \rceil - |x|)$ is less than the tolerance then the integer restrictions assumed to be satisfied.

`Accepted Values:` [1e-9 ;+inf]

`Default Value:` 1.0e-5

`Groups:` *Mixed-integer optimization*

`dparam.mio_tol_feas`
Feasibility tolerance for mixed integer solver.

`Accepted Values:` [1e-9 ;1e-3]

`Default Value:` 1.0e-6

`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.

`Accepted Values:` [0.0 ;1.0]

`Default Value:` 0.0

`Groups:` *Mixed-integer optimization*

`dparam.mio_tol_rel_gap`
Relative optimality tolerance employed by the mixed-integer optimizer.

`Accepted Values:` [0.0 ;+inf]

`Default Value:` 1.0e-4

`Groups:` *Mixed-integer optimization*, *Termination criterion*

`dparam.optimizer_max_time`
Maximum amount of time the optimizer is allowed to spent on the optimization. A negative number means infinity.

`Accepted Values:` [-inf ;+inf]

`Default Value:` -1.0

`Groups:` *Termination criterion*

`dparam.presolve_tol_abs_lindep`
Absolute tolerance employed by the linear dependency checker.

`Accepted Values:` [0.0 ;+inf]

`Default Value:` 1.0e-6

`Groups:` *Presolve*

www.EngineeringBooksPdf.com

`dparam.presolve_tol_aij`
Absolute zero tolerance employed for a_{ij} in the presolve.

`Accepted Values:` [1.0e-15 ;+inf]

`Default Value:` 1.0e-12

`Groups:` *Presolve*

`dparam.presolve_tol_rel_lindep`
Relative tolerance employed by the linear dependency checker.

`Accepted Values:` [0.0 ;+inf]

`Default Value:` 1.0e-10

`Groups:` *Presolve*

`dparam.presolve_tol_s`
Absolute zero tolerance employed for s_i in the presolve.

`Accepted Values:` [0.0 ;+inf]

`Default Value:` 1.0e-8

`Groups:` *Presolve*

`dparam.presolve_tol_x`
Absolute zero tolerance employed for x_j in the presolve.

`Accepted Values:` [0.0 ;+inf]

`Default Value:` 1.0e-8

`Groups:` *Presolve*

`dparam.qcqo_reformulate_rel_drop_tol`
This parameter determines when columns are dropped in incomplete Cholesky factorization during reformulation of quadratic problems.

`Accepted Values:` [0 ;+inf]

`Default Value:` 1e-15

`Groups:` *Interior-point method*

`dparam.semidefinite_tol_approx`
Tolerance to define a matrix to be positive semidefinite.

`Accepted Values:` [1.0e-15 ;+inf]

`Default Value:` 1.0e-10

`Groups:` *Data check*

`dparam.simplex_abs_tol_piv`
Absolute pivot tolerance employed by the simplex optimizers.

`Accepted Values:` [1.0e-12 ;+inf]

`Default Value:` 1.0e-7

`Groups:` *Simplex optimizer*

`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.

`Accepted Values:` [1.0e-6 ;0.999999]

`Default Value:` 0.01

www.EngineeringBooksPdf.com

Groups: *Basis identification*, *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.

  Accepted Values: [-inf ;+inf]

  Default Value: 1.0e30

  Groups: *Termination criterion*

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 ∞.

  Accepted Values: [-inf ;+inf]

  Default Value: 0.5e30

  Groups: *Termination criterion*

Integer Parameters

iparam.ana_sol_basis
: Controls whether the basis matrix is analyzed in solution analyzer.

  Accepted Values: *onoffkey*

  Default Value: *onoffkey.on*

  Groups: *Analysis*

iparam.ana_sol_print_violated
: Controls whether a list of violated constraints is printed when calling *task.analyzesolution*.

  All constraints violated by more than the value set by the parameter *dparam.ana_sol_infeas_tol* will be printed.

  Accepted Values: *onoffkey*

  Default Value: *onoffkey.off*

  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.

  Accepted Values: *onoffkey*

  Default Value: *onoffkey.off*

  Groups: *Debugging*

iparam.auto_update_sol_info
: Controls whether the solution information items are automatically updated after an optimization is performed.

  Accepted Values: *onoffkey*

  Default Value: *onoffkey.off*

  Groups: *Optimization 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.

www.EngineeringBooksPdf.com

This has significance for the results returned by the *task.solvewithbasis* function.

Accepted Values: *onoffkey*

Default Value: *onoffkey.off*

Groups: *Simplex optimizer*

iparam.bi_clean_optimizer
: Controls which simplex optimizer is used in the clean-up phase.

  Accepted Values: *optimizertype*

  Default Value: *optimizertype.free*

  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*.

  Accepted Values: *onoffkey*

  Default Value: *onoffkey.off*

  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*.

  Accepted Values: *onoffkey*

  Default Value: *onoffkey.off*

  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.

  Accepted Values: [0 ;+inf]

  Default Value: 1000000

  Groups: *Basis identification*, *Termination criterion*

iparam.cache_license
: Specifies if the license is kept checked out for the lifetime of the mosek environment (*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.

  Accepted Values: *onoffkey*

  Default Value: *onoffkey.on*

  Groups: *License manager*

iparam.check_convexity
: Specify the level of convexity check on quadratic problems

  Accepted Values: *checkconvexitytype*

  Default Value: *checkconvexitytype.full*

  Groups: *Data check*, *Nonlinear convex method*

www.EngineeringBooksPdf.com

`iparam.compress_statfile`
Control compression of stat files.

Accepted Values: *onoffkey*

Default Value: *onoffkey.on*

`iparam.infeas_generic_names`
Controls whether generic names are used when an infeasible subproblem is created.

Accepted Values: *onoffkey*

Default Value: *onoffkey.off*

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.

Accepted Values: *onoffkey*

Default Value: *onoffkey.on*

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.

Accepted Values: *onoffkey*

Default Value: *onoffkey.off*

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.

Accepted Values: [0 ;+inf]

Default Value: 1

Groups: *Infeasibility report*, *Output information*

`iparam.intpnt_basis`
Controls whether the interior-point optimizer also computes an optimal basis.

Accepted Values: *basindtype*

Default Value: *basindtype.always*

Groups: *Interior-point method*, *Basis identification*

`iparam.intpnt_diff_step`
Controls whether different step sizes are allowed in the primal and dual space.

Accepted Values: *onoffkey*

Default Value: *onoffkey.on*

Groups: *Interior-point method*

`iparam.intpnt_hotstart`
Currently not in use.

Accepted Values: *intpnthotstart*

Default Value: *intpnthotstart.none*

Groups: *Interior-point method*

www.EngineeringBooksPdf.com

`iparam.intpnt_max_iterations`
Controls the maximum number of iterations allowed in the interior-point optimizer.

`Accepted Values:` [0 ;+inf]

`Default Value:` 400

`Groups:` *Interior-point method*, *Termination criterion*

`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.

`Accepted Values:` [-1 ;+inf]

`Default Value:` -1

`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.

`Accepted Values:` [-inf ;+inf]

`Default Value:` -1

`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.

`Accepted Values:` *onoffkey*

`Default Value:` *onoffkey.on*

`Groups:` *Optimization 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

`Accepted Values:` [0 ;+inf]

`Default Value:` 40

`Groups:` *Interior-point method*

`iparam.intpnt_order_method`
Controls the ordering strategy used by the interior-point optimizer when factorizing the Newton equation system.

`Accepted Values:` *orderingtype*

`Default Value:` *orderingtype.free*

`Groups:` *Interior-point method*

`iparam.intpnt_regularization_use`
Controls whether regularization is allowed.

`Accepted Values:` *onoffkey*

`Default Value:` *onoffkey.on*

`Groups:` *Interior-point method*

www.EngineeringBooksPdf.com

`iparam.intpnt_scaling`
Controls how the problem is scaled before the interior-point optimizer is used.

`Accepted Values:` *scalingtype*

`Default Value:` *scalingtype.free*

`Groups:` *Interior-point method*

`iparam.intpnt_solve_form`
Controls whether the primal or the dual problem is solved.

`Accepted Values:` *solveform*

`Default Value:` *solveform.free*

`Groups:` *Interior-point method*

`iparam.intpnt_starting_point`
Starting point used by the interior-point optimizer.

`Accepted Values:` *startpointtype*

`Default Value:` *startpointtype.free*

`Groups:` *Interior-point method*

`iparam.license_debug`
This option is used to turn on debugging of the license manager.

`Accepted Values:` *onoffkey*

`Default Value:` *onoffkey.off*

`Groups:` *License manager*

`iparam.license_pause_time`
If *iparam.license_wait* = *onoffkey.on* and no license is available, then **MOSEK** sleeps a number of milliseconds between each check of whether a license has become free.

`Accepted Values:` [0 ;1000000]

`Default Value:` 100

`Groups:` *License manager*

`iparam.license_suppress_expire_wrns`
Controls whether license features expire warnings are suppressed.

`Accepted Values:` *onoffkey*

`Default Value:` *onoffkey.off*

`Groups:` *License manager*, *Output information*

`iparam.license_trh_expiry_wrn`
If a license feature expires in a numbers days less than the value of this parameter then a warning will be issued.

`Accepted Values:` [0 ;+inf]

`Default Value:` 7

`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.

`Accepted Values:` *onoffkey*

`Default Value:` *onoffkey.off*

`Groups:` *Overall solver*, *Optimization system*, *License manager*

www.EngineeringBooksPdf.com

`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.

  `Accepted Values:` [0 ;+inf]

  `Default Value:` 10

  `Groups:` *Output information*, *Logging*

`iparam.log_ana_pro`
: Controls amount of output from the problem analyzer.

  `Accepted Values:` [0 ;+inf]

  `Default Value:` 1

  `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.

  `Accepted Values:` [0 ;+inf]

  `Default Value:` 4

  `Groups:` *Basis identification*, *Output information*, *Logging*

`iparam.log_bi_freq`
: Controls how frequent the optimizer outputs information about the basis identification and how frequent the user-defined call-back function is called.

  `Accepted Values:` [0 ;+inf]

  `Default Value:` 2500

  `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.

  `Accepted Values:` [0 ;+inf]

  `Default Value:` 0

  `Groups:` *Data check*, *Nonlinear convex method*

`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.

  `Accepted Values:` [0 ;+inf]

  `Default Value:` 1

  `Groups:` *Output information*, *Logging*

`iparam.log_expand`
: Controls the amount of logging when a data item such as the maximum number constrains is expanded.

  `Accepted Values:` [0 ;+inf]

www.EngineeringBooksPdf.com

Default Value: 0

Groups: *Output information*, *Logging*

`iparam.log_factor`
If turned on, then the factor log lines are added to the log.

Accepted Values: [0 ;+inf]

Default Value: 1

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.

Accepted Values: [0 ;+inf]

Default Value: 1

Groups: *Output information*, *Logging*

`iparam.log_file`
If turned on, then some log info is printed when a file is written or read.

Accepted Values: [0 ;+inf]

Default Value: 1

Groups: *Data input/output*, *Output information*, *Logging*

`iparam.log_head`
If turned on, then a header line is added to the log.

Accepted Values: [0 ;+inf]

Default Value: 1

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.

Accepted Values: [0 ;+inf]

Default Value: 1

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.

Accepted Values: [0 ;+inf]

Default Value: 4

Groups: *Interior-point method*, *Output information*, *Logging*

`iparam.log_mio`
Controls the log level for the mixed-integer optimizer. A higher level implies that more information is logged.

Accepted Values: [0 ;+inf]

Default Value: 4

Groups: *Mixed-integer optimization*, *Output information*, *Logging*

www.EngineeringBooksPdf.com

`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.

`Accepted Values:` [-inf ;+inf]

`Default Value:` 10

`Groups:` *Mixed-integer optimization*, *Output information*, *Logging*

`iparam.log_optimizer`
Controls the amount of general optimizer information that is logged.

`Accepted Values:` [0 ;+inf]

`Default Value:` 1

`Groups:` *Output information*, *Logging*

`iparam.log_order`
If turned on, then factor lines are added to the log.

`Accepted Values:` [0 ;+inf]

`Default Value:` 1

`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.

`Accepted Values:` [0 ;+inf]

`Default Value:` 1

`Groups:` *Interior-point method*, *Logging*

`iparam.log_response`
Controls amount of output printed when response codes are reported. A higher level implies that more information is logged.

`Accepted Values:` [0 ;+inf]

`Default Value:` 0

`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.

`Accepted Values:` [0 ;+inf]

`Default Value:` 1

`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.

`Accepted Values:` [0 ;+inf]

`Default Value:` 0

`Groups:` *Output information*, *Logging*

www.EngineeringBooksPdf.com

`iparam.log_sim`
Controls amount of output printed by the simplex optimizer. A higher level implies that more information is logged.

`Accepted Values:` [0 ;+inf]

`Default Value:` 4

`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 call-back function is called.

`Accepted Values:` [0 ;+inf]

`Default Value:` 1000

`Groups:` *Simplex optimizer*, *Output information*, *Logging*

`iparam.log_sim_minor`
Currently not in use.

`Accepted Values:` [0 ;+inf]

`Default Value:` 1

`Groups:` *Simplex optimizer*, *Output information*

`iparam.log_storage`
When turned on, **MOSEK** prints messages regarding the storage usage and allocation.

`Accepted Values:` [0 ;+inf]

`Default Value:` 0

`Groups:` *Output information*, *Optimization system*, *Logging*

`iparam.max_num_warnings`
Each warning is shown a limit number times controlled by this parameter. A negative value is identical to infinite number of times.

`Accepted Values:` [-inf ;+inf]

`Default Value:` 10

`Groups:` *Output information*

`iparam.mio_branch_dir`
Controls whether the mixed-integer optimizer is branching up or down by default.

`Accepted Values:` *branchdir*

`Default Value:` *branchdir.free*

`Groups:` *Mixed-integer optimization*

`iparam.mio_construct_sol`
If set to *onoffkey.on* and all integer variables have been given a value for which a feasible mixed integer solution exists, then **MOSEK** generates an initial solution to the mixed integer problem by fixing all integer values and solving the remaining problem.

`Accepted Values:` *onoffkey*

`Default Value:` *onoffkey.off*

`Groups:` *Mixed-integer optimization*

`iparam.mio_cut_clique`
Controls whether clique cuts should be generated.

`Accepted Values:` *onoffkey*

www.EngineeringBooksPdf.com

Default Value: *onoffkey.on*

Groups: *Mixed-integer optimization*

`iparam.mio_cut_cmir`
Controls whether mixed integer rounding cuts should be generated.

Accepted Values: *onoffkey*

Default Value: *onoffkey.on*

Groups: *Mixed-integer optimization*

`iparam.mio_cut_gmi`
Controls whether GMI cuts should be generated.

Accepted Values: *onoffkey*

Default Value: *onoffkey.on*

Groups: *Mixed-integer optimization*

`iparam.mio_cut_implied_bound`
Controls whether implied bound cuts should be generated.

Accepted Values: *onoffkey*

Default Value: *onoffkey.off*

Groups: *Mixed-integer optimization*

`iparam.mio_cut_knapsack_cover`
Controls whether knapsack cover cuts should be generated.

Accepted Values: *onoffkey*

Default Value: *onoffkey.off*

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

Accepted Values: [-1 ;+1]

Default Value: -1

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.

Accepted Values: [-inf ;+inf]

Default Value: -1

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.

Accepted Values: [-inf ;+inf]

www.EngineeringBooksPdf.com

Default Value: -1

Groups: *Mixed-integer optimization*, *Termination criterion*

`iparam.mio_max_num_relaxs`
Maximum number of relaxations allowed during the branch and bound search. A negative value means infinite.

Accepted Values: [-inf ;+inf]

Default Value: -1

Groups: *Mixed-integer optimization*

`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.

Accepted Values: [-inf ;+inf]

Default Value: -1

Groups: *Mixed-integer optimization*, *Termination criterion*

`iparam.mio_mode`
Controls whether the optimizer includes the integer restrictions when solving a (mixed) integer optimization problem.

Accepted Values: *miomode*

Default Value: *miomode.satisfied*

Groups: *Overall solver*

`iparam.mio_mt_user_cb`
It true user callbacks are called from each thread used by this optimizer. If false the user callback is only called from a single thread.

Accepted Values: *onoffkey*

Default Value: *onoffkey.off*

Groups: *Optimization system*

`iparam.mio_node_optimizer`
Controls which optimizer is employed at the non-root nodes in the mixed-integer optimizer.

Accepted Values: *optimizertype*

Default Value: *optimizertype.free*

Groups: *Mixed-integer optimization*

`iparam.mio_node_selection`
Controls the node selection strategy employed by the mixed-integer optimizer.

Accepted Values: *mionodeseltype*

Default Value: *mionodeseltype.free*

Groups: *Mixed-integer optimization*

`iparam.mio_perspective_reformulate`
Enables or disables perspective reformulation in presolve.

Accepted Values: *onoffkey*

Default Value: *onoffkey.on*

Groups: *Mixed-integer optimization*

www.EngineeringBooksPdf.com

`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

`Accepted Values:` [-inf ;+inf]

`Default Value:` -1

`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.

`Accepted Values:` [-1 ;+inf]

`Default Value:` -1

`Groups:` *Mixed-integer optimization*

`iparam.mio_root_optimizer`
Controls which optimizer is employed at the root node in the mixed-integer optimizer.

`Accepted Values:` *optimizertype*

`Default Value:` *optimizertype.free*

`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

`Accepted Values:` [-1 ;1]

`Default Value:` -1

`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

`Accepted Values:` [-1 ;+2]

`Default Value:` -1

`Groups:` *Mixed-integer optimization*

`iparam.mt_spincount`
Set the number of iterations to spin before sleeping.

`Accepted Values:` [0 ;1000000000]

www.EngineeringBooksPdf.com

Default Value: 0

Groups: *Optimization 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.

Accepted Values: [0 ;+inf]

Default Value: 0

Groups: *Optimization system*

`iparam.opf_max_terms_per_line`
The maximum number of terms (linear and quadratic) per line when an OPF file is written.

Accepted Values: [0 ;+inf]

Default Value: 5

Groups: *Data input/output*

`iparam.opf_write_header`
Write a text header with date and **MOSEK** version in an OPF file.

Accepted Values: *onoffkey*

Default Value: *onoffkey.on*

Groups: *Data input/output*

`iparam.opf_write_hints`
Write a hint section with problem dimensions in the beginning of an OPF file.

Accepted Values: *onoffkey*

Default Value: *onoffkey.on*

Groups: *Data input/output*

`iparam.opf_write_parameters`
Write a parameter section in an OPF file.

Accepted Values: *onoffkey*

Default Value: *onoffkey.off*

Groups: *Data input/output*

`iparam.opf_write_problem`
Write objective, constraints, bounds etc. to an OPF file.

Accepted Values: *onoffkey*

Default Value: *onoffkey.on*

Groups: *Data input/output*

`iparam.opf_write_solutions`
Enable inclusion of solutions in the OPF files.

Accepted Values: *onoffkey*

Default Value: *onoffkey.off*

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.

Accepted Values: *onoffkey*

www.EngineeringBooksPdf.com

Default Value: *onoffkey.on*

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.

Accepted Values: *onoffkey*

Default Value: *onoffkey.on*

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.

Accepted Values: *onoffkey*

Default Value: *onoffkey.on*

Groups: *Data input/output*

iparam.optimizer
The parameter controls which optimizer is used to optimize the task.

Accepted Values: *optimizertype*

Default Value: *optimizertype.free*

Groups: *Overall solver*

iparam.param_read_case_name
If turned on, then names in the parameter file are case sensitive.

Accepted Values: *onoffkey*

Default Value: *onoffkey.on*

Groups: *Data input/output*

iparam.param_read_ign_error
If turned on, then errors in parameter settings is ignored.

Accepted Values: *onoffkey*

Default Value: *onoffkey.off*

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 menas the parameter value is selected automatically.

Accepted Values: [-inf ;+inf]

Default Value: -1

Groups: *Presolve*

iparam.presolve_eliminator_max_num_tries
Control the maximum number of times the eliminator is tried.

Accepted Values: [-inf ;+inf]

Default Value: -1

Groups: *Presolve*

iparam.presolve_level
Currently not used.

Accepted Values: [-inf ;+inf]

www.EngineeringBooksPdf.com

Default Value: -1

Groups: *Overall solver*, *Presolve*

`iparam.presolve_lindep_abs_work_trh`
The linear dependency check is potentially computationally expensive.

Accepted Values: [-inf ;+inf]

Default Value: 100

Groups: *Presolve*

`iparam.presolve_lindep_rel_work_trh`
The linear dependency check is potentially computationally expensive.

Accepted Values: [-inf ;+inf]

Default Value: 100

Groups: *Presolve*

`iparam.presolve_lindep_use`
Controls whether the linear constraints are checked for linear dependencies.

Accepted Values: *onoffkey*

Default Value: *onoffkey.on*

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.

Accepted Values: [-inf ;+inf]

Default Value: -1

`iparam.presolve_use`
Controls whether the presolve is applied to a problem before it is optimized.

Accepted Values: *presolvemode*

Default Value: *presolvemode.free*

Groups: *Overall solver*, *Presolve*

`iparam.primal_repair_optimizer`
Controls which optimizer that is used to find the optimal repair.

Accepted Values: *optimizertype*

Default Value: *optimizertype.free*

Groups: *Overall solver*

`iparam.read_data_compressed`
If this option is turned on,it is assumed that the data file is compressed.

Accepted Values: *compresstype*

Default Value: *compresstype.free*

Groups: *Data input/output*

`iparam.read_data_format`
Format of the data file to be read.

Accepted Values: *dataformat*

Default Value: *dataformat.extension*

www.EngineeringBooksPdf.com

Groups: *Data input/output*

`iparam.read_debug`
Turns on additional debugging information when reading files.

Accepted Values: *onoffkey*

Default Value: *onoffkey.off*

Groups: *Data input/output*

`iparam.read_keep_free_con`
Controls whether the free constraints are included in the problem.

Accepted Values: *onoffkey*

Default Value: *onoffkey.off*

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.

Accepted Values: *onoffkey*

Default Value: *onoffkey.off*

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.

Accepted Values: *onoffkey*

Default Value: *onoffkey.on*

Groups: *Data input/output*

`iparam.read_mps_format`
Controls how strictly the MPS file reader interprets the MPS format.

Accepted Values: *mpsformat*

Default Value: *mpsformat.free*

Groups: *Data input/output*

`iparam.read_mps_width`
Controls the maximal number of characters allowed in one line of the MPS file.

Accepted Values: [80 ;+inf]

Default Value: 1024

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.

Accepted Values: *onoffkey*

Default Value: *onoffkey.off*

Groups: *Data input/output*

`iparam.sensitivity_all`
If set to *onoffkey.on*, then *task.sensitivityreport* analyzes all bounds and variables instead of reading a specification from the file.

Accepted Values: *onoffkey*

Default Value: *onoffkey.off*

www.EngineeringBooksPdf.com

Groups: *Overall solver*

`iparam.sensitivity_optimizer`
Controls which optimizer is used for optimal partition sensitivity analysis.

Accepted Values: *optimizertype*

Default Value: *optimizertype.free_simplex*

Groups: *Overall solver*, *Simplex optimizer*

`iparam.sensitivity_type`
Controls which type of sensitivity analysis is to be performed.

Accepted Values: *sensitivitytype*

Default Value: *sensitivitytype.basis*

Groups: *Overall solver*

`iparam.sim_basis_factor_use`
Controls whether a (LU) factorization of the basis is used in a hot-start. Forcing a refactorization sometimes improves the stability of the simplex optimizers, but in most cases there is a performance penalty.

Accepted Values: *onoffkey*

Default Value: *onoffkey.on*

Groups: *Simplex optimizer*

`iparam.sim_degen`
Controls how aggressively degeneration is handled.

Accepted Values: *simdegen*

Default Value: *simdegen.free*

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.

Accepted Values: [0 ;+inf]

Default Value: 90

Groups: *Dual simplex optimizer*

`iparam.sim_dual_phaseone_method`
An experimental feature.

Accepted Values: [0 ;10]

Default Value: 0

Groups: *Simplex optimizer*

`iparam.sim_dual_restrict_selection`
The dual simplex optimizer can use a so-called restricted selection/pricing strategy to chooses 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.

Accepted Values: [0 ;100]

Default Value: 50

www.EngineeringBooksPdf.com

Groups: *Dual simplex optimizer*

`iparam.sim_dual_selection`
Controls the choice of the incoming variable, known as the selection strategy, in the dual simplex optimizer.

Accepted Values: *simseltype*

Default Value: *simseltype.free*

Groups: *Dual simplex optimizer*

`iparam.sim_exploit_dupvec`
Controls if the simplex optimizers are allowed to exploit duplicated columns.

Accepted Values: *simdupvec*

Default Value: *simdupvec.off*

Groups: *Simplex optimizer*

`iparam.sim_hotstart`
Controls the type of hot-start that the simplex optimizer perform.

Accepted Values: *simhotstart*

Default Value: *simhotstart.free*

Groups: *Simplex optimizer*

`iparam.sim_hotstart_lu`
Determines if the simplex optimizer should exploit the initial factorization.

Accepted Values: *onoffkey*

Default Value: *onoffkey.on*

`iparam.sim_integer`
An experimental feature.

Accepted Values: [0 ;10]

Default Value: 0

Groups: *Simplex optimizer*

`iparam.sim_max_iterations`
Maximum number of iterations that can be used by a simplex optimizer.

Accepted Values: [0 ;+inf]

Default Value: 10000000

Groups: *Simplex optimizer*, *Termination criterion*

`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.

Accepted Values: [0 ;+inf]

Default Value: 250

Groups: *Simplex optimizer*

`iparam.sim_non_singular`
Controls if the simplex optimizer ensures a non-singular basis, if possible.

Accepted Values: *onoffkey*

Default Value: *onoffkey.on*

Groups: *Simplex optimizer*

www.EngineeringBooksPdf.com

`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.

`Accepted Values:` [0 ;+inf]

`Default Value:` 90

`Groups:` *Primal simplex optimizer*

`iparam.sim_primal_phaseone_method`
An experimental feature.

`Accepted Values:` [0 ;10]

`Default Value:` 0

`Groups:` *Simplex optimizer*

`iparam.sim_primal_restrict_selection`
The primal simplex optimizer can use a so-called restricted selection/pricing strategy to chooses the outgoing variable. Hence, if restricted selection is applied, then the primal simplex optimizer first choose a subset of all the potential incoming variables. Next, for some time it will choose the incoming variable only among the subset. From time to time the subset is redefined.

A larger value of this parameter implies that the optimizer will be more aggressive in its restriction strategy, i.e. a value of 0 implies that the restriction strategy is not applied at all.

`Accepted Values:` [0 ;100]

`Default Value:` 50

`Groups:` *Primal simplex optimizer*

`iparam.sim_primal_selection`
Controls the choice of the incoming variable, known as the selection strategy, in the primal simplex optimizer.

`Accepted Values:` *simseltype*

`Default Value:` *simseltype.free*

`Groups:` *Primal simplex optimizer*

`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.

`Accepted Values:` [0 ;+inf]

`Default Value:` 0

`Groups:` *Simplex optimizer*

`iparam.sim_reformulation`
Controls if the simplex optimizers are allowed to reformulate the problem.

`Accepted Values:` *simreform*

`Default Value:` *simreform.off*

`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.

`Accepted Values:` *onoffkey*

www.EngineeringBooksPdf.com

Default Value: *onoffkey.off*

Groups: *Simplex optimizer*

`iparam.sim_scaling`
Controls how much effort is used in scaling the problem before a simplex optimizer is used.

Accepted Values: *scalingtype*

Default Value: *scalingtype.free*

Groups: *Simplex optimizer*

`iparam.sim_scaling_method`
Controls how the problem is scaled before a simplex optimizer is used.

Accepted Values: *scalingmethod*

Default Value: *scalingmethod.pow2*

Groups: *Simplex optimizer*

`iparam.sim_solve_form`
Controls whether the primal or the dual problem is solved by the primal-/dual-simplex optimizer.

Accepted Values: *solveform*

Default Value: *solveform.free*

Groups: *Simplex optimizer*

`iparam.sim_stability_priority`
Controls how high priority the numerical stability should be given.

Accepted Values: [0 ;100]

Default Value: 50

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).

Accepted Values: *onoffkey*

Default Value: *onoffkey.off*

Groups: *Simplex optimizer*

`iparam.solution_callback`
Indicates whether solution call-backs will be performed during the optimization.

Accepted Values: *onoffkey*

Default Value: *onoffkey.off*

Groups: *Progress call-back*, *Overall solver*

`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.

Accepted Values: *onoffkey*

Default Value: *onoffkey.off*

Groups: *Solution input/output*

www.EngineeringBooksPdf.com

`iparam.sol_filter_keep_ranged`
If turned on, then ranged constraints and variables are written to the solution file independent of the filter setting.

`Accepted Values:` *onoffkey*

`Default Value:` *onoffkey.off*

`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.

`Accepted Values:` [-inf ;+inf]

`Default Value:` -1

`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**.

`Accepted Values:` [0 ;+inf]

`Default Value:` 1024

`Groups:` *Data input/output*, *Solution input/output*

`iparam.timing_level`
Controls the a amount of timing performed inside **MOSEK**.

`Accepted Values:` [0 ;+inf]

`Default Value:` 1

`Groups:` *Optimization system*

`iparam.write_bas_constraints`
Controls whether the constraint section is written to the basic solution file.

`Accepted Values:` *onoffkey*

`Default Value:` *onoffkey.on*

`Groups:` *Data input/output*, *Solution input/output*

`iparam.write_bas_head`
Controls whether the header section is written to the basic solution file.

`Accepted Values:` *onoffkey*

`Default Value:` *onoffkey.on*

`Groups:` *Data input/output*, *Solution input/output*

`iparam.write_bas_variables`
Controls whether the variables section is written to the basic solution file.

`Accepted Values:` *onoffkey*

`Default Value:` *onoffkey.on*

`Groups:` *Data input/output*, *Solution input/output*

`iparam.write_data_compressed`
Controls whether the data file is compressed while it is written. 0 means no compression while higher values mean more compression.

`Accepted Values:` [0 ;+inf]

`Default Value:` 0

www.EngineeringBooksPdf.com

Groups: *Data input/output*

`iparam.write_data_format`
Controls the data format when a task is written using *task.writedata*.

Accepted Values: *dataformat*

Default Value: *dataformat.extension*

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.

Accepted Values: *onoffkey*

Default Value: *onoffkey.off*

Groups: *Data input/output*

`iparam.write_free_con`
Controls whether the free constraints are written to the data file.

Accepted Values: *onoffkey*

Default Value: *onoffkey.on*

Groups: *Data input/output*

`iparam.write_generic_names`
Controls whether the generic names or user-defined names are used in the data file.

Accepted Values: *onoffkey*

Default Value: *onoffkey.off*

Groups: *Data input/output*

`iparam.write_generic_names_io`
Index origin used in generic names.

Accepted Values: [0 ;+inf]

Default Value: 1

Groups: *Data input/output*

`iparam.write_ignore_incompatible_items`
Controls if the writer ignores incompatible problem items when writing files.

Accepted Values: *onoffkey*

Default Value: *onoffkey.off*

Groups: *Data input/output*

`iparam.write_int_constraints`
Controls whether the constraint section is written to the integer solution file.

Accepted Values: *onoffkey*

Default Value: *onoffkey.on*

Groups: *Data input/output*, *Solution input/output*

`iparam.write_int_head`
Controls whether the header section is written to the integer solution file.

Accepted Values: *onoffkey*

Default Value: *onoffkey.on*

Groups: *Data input/output*, *Solution input/output*

www.EngineeringBooksPdf.com

`iparam.write_int_variables`
Controls whether the variables section is written to the integer solution file.

`Accepted Values:` *onoffkey*

`Default Value:` *onoffkey.on*

`Groups:` *Data input/output*, *Solution input/output*

`iparam.write_lp_full_obj`
Write all variables, including the ones with 0-coefficients, in the objective.

`Accepted Values:` *onoffkey*

`Default Value:` *onoffkey.on*

`Groups:` *Data input/output*

`iparam.write_lp_line_width`
Maximum width of line in an LP file written by **MOSEK**.

`Accepted Values:` [40 ;+inf]

`Default Value:` 80

`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.

`Accepted Values:` *onoffkey*

`Default Value:` *onoffkey.on*

`Groups:` *Data input/output*

`iparam.write_lp_strict_format`
Controls whether LP output files satisfy the LP format strictly.

`Accepted Values:` *onoffkey*

`Default Value:` *onoffkey.off*

`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.

`Accepted Values:` [0 ;+inf]

`Default Value:` 10

`Groups:` *Data input/output*

`iparam.write_mps_format`
Controls in which format the MPS is written.

`Accepted Values:` *mpsformat*

`Default Value:` *mpsformat.free*

`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.

`Accepted Values:` *onoffkey*

`Default Value:` *onoffkey.on*

`Groups:` *Data input/output*

www.EngineeringBooksPdf.com

`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.

`Accepted Values:` [0 ;+inf]

`Default Value:` 15

`Groups:` *Data input/output*

`iparam.write_sol_barvariables`
Controls whether the symmetric matrix variables section is written to the solution file.

`Accepted Values:` *onoffkey*

`Default Value:` *onoffkey.on*

`Groups:` *Data input/output*, *Solution input/output*

`iparam.write_sol_constraints`
Controls whether the constraint section is written to the solution file.

`Accepted Values:` *onoffkey*

`Default Value:` *onoffkey.on*

`Groups:` *Data input/output*, *Solution input/output*

`iparam.write_sol_head`
Controls whether the header section is written to the solution file.

`Accepted Values:` *onoffkey*

`Default Value:` *onoffkey.on*

`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.

`Accepted Values:` *onoffkey*

`Default Value:` *onoffkey.off*

`Groups:` *Data input/output*, *Solution input/output*

`iparam.write_sol_variables`
Controls whether the variables section is written to the solution file.

`Accepted Values:` *onoffkey*

`Default Value:` *onoffkey.on*

`Groups:` *Data input/output*, *Solution input/output*

`iparam.write_task_inc_sol`
Controls whether the solutions are stored in the task file too.

`Accepted Values:` *onoffkey*

`Default Value:` *onoffkey.on*

`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.

`Accepted Values:` *xmlwriteroutputtype*

`Default Value:` *xmlwriteroutputtype.row*

`Groups:` *Data input/output*

www.EngineeringBooksPdf.com

String Parameters

`sparam.bas_sol_file_name`
Name of the `bas` solution file.

`Accepted Values:` Any valid file name.

`Groups:` *Data input/output*, *Solution input/output*

`sparam.data_file_name`
Data are read and written to this file.

`Accepted Values:` Any valid file name.

`Groups:` *Data input/output*

`sparam.debug_file_name`
MOSEK debug file.

`Accepted Values:` Any valid file name.

`Groups:` *Data input/output*

`sparam.int_sol_file_name`
Name of the `int` solution file.

`Accepted Values:` Any valid file name.

`Groups:` *Data input/output*, *Solution input/output*

`sparam.itr_sol_file_name`
Name of the `itr` solution file.

`Accepted Values:` Any valid file name.

`Groups:` *Data input/output*, *Solution input/output*

`sparam.mio_debug_string`
For internal use only.

`Accepted Values:` Any valid 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.

`Accepted Values:` Any valid string.

`Default Value:` %%

`Groups:` *Data input/output*

`sparam.param_read_file_name`
Modifications to the parameter database is read from this file.

`Accepted Values:` Any valid file name.

`Groups:` *Data input/output*

`sparam.param_write_file_name`
The parameter database is written to this file.

`Accepted Values:` Any valid 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 Values:` Any valid MPS name.

www.EngineeringBooksPdf.com

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 Values: Any valid MPS 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 Values: Any valid MPS 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 Values: Any valid MPS 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 Values: Any valid string.

`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 Values: Any valid string.

Groups: *Data input/output*

`sparam.sensitivity_res_file_name`
If this is a nonempty string, then *task.sensitivityreport* writes results to this file.

Accepted Values: Any valid string.

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 Values: Any valid filter.

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 Values: Any valid filter.

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 Values: Any valid filter.

Groups: *Data input/output*, *Solution input/output*

www.EngineeringBooksPdf.com

`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 Values:` Any valid file name.

`Groups:` *Data input/output*, *Solution input/output*

`sparam.stat_file_name`
Statistics file name.

`Accepted Values:` Any valid file name.

`Groups:` *Data input/output*

`sparam.stat_key`
Key used when writing the summary file.

`Accepted Values:` Any valid XML string.

`Groups:` *Data input/output*

`sparam.stat_name`
Name used when writing the statistics file.

`Accepted Values:` Any valid XML string.

`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.

`Accepted Values:` Any valid string.

`Default Value:` xmskgen

`Groups:` *Data input/output*

16.4.2 Overall solver parameters.

- *iparam.bi_clean_optimizer*
- *iparam.infeas_prefer_primal*
- *iparam.license_wait*
- *iparam.mio_mode*
- *iparam.optimizer*
- *iparam.presolve_level*
- *iparam.presolve_use*
- *iparam.primal_repair_optimizer*
- *iparam.sensitivity_all*
- *iparam.sensitivity_optimizer*
- *iparam.sensitivity_type*
- *iparam.solution_callback*

www.EngineeringBooksPdf.com

16.4.3 Mixed-integer optimization parameters.

- *iparam.log_mio*
- *iparam.log_mio_freq*
- *iparam.mio_branch_dir*
- *iparam.mio_construct_sol*
- *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*
- *dparam.mio_disable_term_time*
- *iparam.mio_heuristic_level*
- *iparam.mio_max_num_branches*
- *iparam.mio_max_num_relaxs*
- *iparam.mio_max_num_solutions*
- *dparam.mio_max_time*
- *dparam.mio_near_tol_abs_gap*
- *dparam.mio_near_tol_rel_gap*
- *iparam.mio_node_optimizer*
- *iparam.mio_node_selection*
- *iparam.mio_perspective_reformulate*
- *iparam.mio_probing_level*
- *dparam.mio_rel_gap_const*
- *iparam.mio_rins_max_nodes*
- *iparam.mio_root_optimizer*
- *iparam.mio_root_repeat_presolve_level*
- *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.mio_vb_detection_level*

16.4.4 Output information parameters.

- *iparam.infeas_report_level*
- *iparam.license_suppress_expire_wrns*
- *iparam.log*

www.EngineeringBooksPdf.com

- *iparam.log_bi*
- *iparam.log_bi_freq*
- *iparam.log_cut_second_opt*
- *iparam.log_expand*
- *iparam.log_factor*
- *iparam.log_feas_repair*
- *iparam.log_file*
- *iparam.log_head*
- *iparam.log_infeas_ana*
- *iparam.log_intpnt*
- *iparam.log_mio*
- *iparam.log_mio_freq*
- *iparam.log_optimizer*
- *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*

16.4.5 Optimization system parameters.

- *iparam.auto_update_sol_info*
- *iparam.intpnt_multi_thread*
- *iparam.license_wait*
- *iparam.log_storage*
- *iparam.mio_mt_user_cb*
- *iparam.mt_spincount*
- *iparam.num_threads*
- *iparam.timing_level*

16.4.6 Simplex optimizer parameters.

- *dparam.basis_rel_tol_s*
- *iparam.basis_solve_use_plus_one*
- *dparam.basis_tol_s*
- *dparam.basis_tol_x*

www.EngineeringBooksPdf.com

- *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_integer*
- *dparam.sim_lu_tol_rel_piv*
- *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_solve_form*
- *iparam.sim_stability_priority*
- *iparam.sim_switch_optimizer*
- *dparam.simplex_abs_tol_piv*

16.4.7 Logging parameters.

- *iparam.log*
- *iparam.log_ana_pro*
- *iparam.log_bi*
- *iparam.log_bi_freq*
- *iparam.log_cut_second_opt*
- *iparam.log_expand*
- *iparam.log_factor*
- *iparam.log_feas_repair*
- *iparam.log_file*
- *iparam.log_head*
- *iparam.log_infeas_ana*
- *iparam.log_intpnt*
- *iparam.log_mio*

www.EngineeringBooksPdf.com

- *iparam.log_mio_freq*
- *iparam.log_optimizer*
- *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*

16.4.8 Data check parameters.

- *iparam.check_convexity*
- *dparam.data_sym_mat_tol*
- *dparam.data_sym_mat_tol_huge*
- *dparam.data_sym_mat_tol_large*
- *dparam.data_tol_aij*
- *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*
- *iparam.log_check_convexity*
- *dparam.semidefinite_tol_approx*

16.4.9 Progress call-back parameters.

- *iparam.solution_callback*

16.4.10 Analysis parameters.

- *iparam.ana_sol_basis*
- *dparam.ana_sol_infeas_tol*
- *iparam.ana_sol_print_violated*
- *iparam.log_ana_pro*

www.EngineeringBooksPdf.com

16.4.11 Conic interior-point method parameters.

- *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*

16.4.12 Nonlinear convex method parameters.

- *iparam.check_convexity*
- *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.intpnt_tol_infeas*
- *iparam.log_check_convexity*

16.4.13 Solution input/output parameters.

- *sparam.bas_sol_file_name*
- *iparam.infeas_report_auto*
- *sparam.int_sol_file_name*
- *sparam.itr_sol_file_name*
- *iparam.sol_filter_keep_basic*
- *iparam.sol_filter_keep_ranged*
- *sparam.sol_filter_xc_low*
- *sparam.sol_filter_xc_upr*
- *sparam.sol_filter_xx_low*
- *sparam.sol_filter_xx_upr*
- *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*

www.EngineeringBooksPdf.com

- *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*

16.4.14 Interior-point method parameters.

- *iparam.bi_ignore_max_iter*
- *iparam.bi_ignore_num_error*
- *dparam.check_convexity_rel_tol*
- *iparam.intpnt_basis*
- *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*
- *iparam.intpnt_diff_step*
- *iparam.intpnt_hotstart*
- *iparam.intpnt_max_iterations*
- *iparam.intpnt_max_num_cor*
- *iparam.intpnt_max_num_refinement_steps*
- *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*
- *iparam.intpnt_off_col_trh*
- *iparam.intpnt_order_method*
- *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*
- *iparam.intpnt_regularization_use*

www.EngineeringBooksPdf.com

- *iparam.intpnt_scaling*
- *iparam.intpnt_solve_form*
- *iparam.intpnt_starting_point*
- *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*
- *iparam.log_intpnt*
- *iparam.log_presolve*
- *dparam.qcqo_reformulate_rel_drop_tol*

16.4.15 Dual simplex optimizer parameters.

- *iparam.sim_dual_crash*
- *iparam.sim_dual_restrict_selection*
- *iparam.sim_dual_selection*

16.4.16 Termination criterion parameters.

- *dparam.basis_rel_tol_s*
- *dparam.basis_tol_s*
- *dparam.basis_tol_x*
- *iparam.bi_max_iterations*
- *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*
- *iparam.intpnt_max_iterations*
- *dparam.intpnt_nl_tol_dfeas*
- *dparam.intpnt_nl_tol_mu_red*
- *dparam.intpnt_nl_tol_near_rel*
- *dparam.intpnt_nl_tol_pfeas*

www.EngineeringBooksPdf.com

- *dparam.intpnt_nl_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_disable_term_time*
- *iparam.mio_max_num_branches*
- *iparam.mio_max_num_solutions*
- *dparam.mio_max_time*
- *dparam.mio_near_tol_rel_gap*
- *dparam.mio_rel_gap_const*
- *dparam.mio_tol_rel_gap*
- *dparam.optimizer_max_time*
- *iparam.sim_max_iterations*
- *dparam.upper_obj_cut*
- *dparam.upper_obj_cut_finite_trh*

16.4.17 Primal simplex optimizer parameters.

- *iparam.sim_primal_crash*
- *iparam.sim_primal_restrict_selection*
- *iparam.sim_primal_selection*

16.4.18 Basis identification parameters.

- *iparam.bi_clean_optimizer*
- *iparam.bi_ignore_max_iter*
- *iparam.bi_ignore_num_error*
- *iparam.bi_max_iterations*
- *iparam.intpnt_basis*
- *iparam.log_bi*

www.EngineeringBooksPdf.com

- *iparam.log_bi_freq*
- *dparam.sim_lu_tol_rel_piv*

16.4.19 License manager parameters.

- *iparam.cache_license*
- *iparam.license_debug*
- *iparam.license_pause_time*
- *iparam.license_suppress_expire_wrns*
- *iparam.license_wait*

16.4.20 Presolve parameters.

- *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*
- *dparam.presolve_tol_abs_lindep*
- *dparam.presolve_tol_aij*
- *dparam.presolve_tol_rel_lindep*
- *dparam.presolve_tol_s*
- *dparam.presolve_tol_x*
- *iparam.presolve_use*

16.4.21 Debugging parameters.

- *iparam.auto_sort_a_before_opt*

16.4.22 Data input/output parameters.

- *sparam.bas_sol_file_name*
- *sparam.data_file_name*
- *sparam.debug_file_name*
- *iparam.infeas_report_auto*
- *sparam.int_sol_file_name*
- *sparam.itr_sol_file_name*
- *iparam.log_file*
- *sparam.mio_debug_string*
- *iparam.opf_max_terms_per_line*
- *iparam.opf_write_header*

www.EngineeringBooksPdf.com

- *iparam.opf_write_hints*
- *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*
- *sparam.param_comment_sign*
- *iparam.param_read_case_name*
- *sparam.param_read_file_name*
- *iparam.param_read_ign_error*
- *sparam.param_write_file_name*
- *iparam.read_data_compressed*
- *iparam.read_data_format*
- *iparam.read_debug*
- *iparam.read_keep_free_con*
- *iparam.read_lp_drop_new_vars_in_bou*
- *iparam.read_lp_quoted_names*
- *sparam.read_mps_bou_name*
- *iparam.read_mps_format*
- *sparam.read_mps_obj_name*
- *sparam.read_mps_ran_name*
- *sparam.read_mps_rhs_name*
- *iparam.read_mps_width*
- *iparam.read_task_ignore_param*
- *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*
- *iparam.sol_read_name_width*
- *iparam.sol_read_width*
- *sparam.stat_file_name*
- *sparam.stat_key*
- *sparam.stat_name*
- *iparam.write_bas_constraints*
- *iparam.write_bas_head*
- *iparam.write_bas_variables*

www.EngineeringBooksPdf.com

- *iparam.write_data_compressed*
- *iparam.write_data_format*
- *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*
- *sparam.write_lp_gen_var_name*
- *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*

16.4.23 Infeasibility report parameters.

- *iparam.infeas_generic_names*
- *iparam.infeas_report_level*
- *iparam.log_infeas_ana*

16.5 Response codes

- *Termination codes*
- *Error codes*
- *Warning codes*

www.EngineeringBooksPdf.com

16.5.1 Termination Codes

`rescode.ok (0)`
No error occurred.

`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.

`rescode.trm_max_iterations (10000)`
The optimizer terminated at the maximum number of iterations.

`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_max_time (10001)`
The optimizer terminated at the maximum amount of time.

`rescode.trm_mio_near_abs_gap (10004)`
The mixed-integer optimizer terminated because the near optimal absolute gap tolerance was satisfied.

`rescode.trm_mio_near_rel_gap (10003)`
The mixed-integer optimizer terminated because the near optimal relative gap tolerance was satisfied.

`rescode.trm_mio_num_branches (10009)`
The mixed-integer optimizer terminated as to the maximum number of branches was reached.

`rescode.trm_mio_num_relaxs (10008)`
The mixed-integer optimizer terminated as the maximum number of relaxations 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_numerical_problem (10025)`
The optimizer terminated due to numerical problems.

`rescode.trm_objective_range (10002)`
The optimizer terminated on the bound of the objective range.

`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 make 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 (near) feasible or near 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 then solution. If the solution near 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 and c) a non-convex problems. Case c) is only relevant for general non-linear problems. It is not possible in general for **MOSEK** to check if a specific problems is convex since such a check would be NP hard in itself. This implies that care should be taken when solving problems involving general user defined functions.

`rescode.trm_user_callback (10007)`
The optimizer terminated due to the return of the user-defined call-back function.

www.EngineeringBooksPdf.com

16.5.2 Error Codes

`rescode.err_ad_invalid_codelist (3102)`
The code list data was invalid.

`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_arg_is_too_large (1227)`
The value of a argument is too small.

`rescode.err_arg_is_too_small (1226)`
The value of a argument is too small.

`rescode.err_argument_dimension (1201)`
A function argument is of incorrect dimension.

`rescode.err_argument_is_too_large (5005)`
The value of a function argument is too large.

`rescode.err_argument_lenneq (1197)`
Incorrect length of arguments.

`rescode.err_argument_perm_array (1299)`
An invalid permutation array is specified.

`rescode.err_argument_type (1198)`
Incorrect argument type.

`rescode.err_bar_var_dim (3920)`
The dimension of a symmetric matrix variable has to greater than 0.

`rescode.err_basis (1266)`
An invalid basis is specified. Either too many or too few basis variables are specified.

`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_blank_name (1070)`
An all blank name has been specified.

`rescode.err_cannot_clone_nl (2505)`
A task with a nonlinear function call-back cannot be cloned.

`rescode.err_cannot_handle_nl (2506)`
A function cannot handle a task with nonlinear function call-backs.

`rescode.err_cbf_duplicate_acoord (7116)`
Duplicate index in ACOORD.

`rescode.err_cbf_duplicate_bcoord (7115)`
Duplicate index in BCOORD.

`rescode.err_cbf_duplicate_con (7108)`
Duplicate CON keyword.

www.EngineeringBooksPdf.com

`rescode.err_cbf_duplicate_int (7110)`
Duplicate INT keyword.

`rescode.err_cbf_duplicate_obj (7107)`
Duplicate OBJ keyword.

`rescode.err_cbf_duplicate_objacoord (7114)`
Duplicate index in OBJCOORD.

`rescode.err_cbf_duplicate_var (7109)`
Duplicate VAR keyword.

`rescode.err_cbf_invalid_con_type (7112)`
Invalid constraint type.

`rescode.err_cbf_invalid_domain_dimension (7113)`
Invalid domain dimension.

`rescode.err_cbf_invalid_int_index (7121)`
Invalid INT index.

`rescode.err_cbf_invalid_var_type (7111)`
Invalid variable type.

`rescode.err_cbf_no_variables (7102)`
No variables are specified.

`rescode.err_cbf_no_version_specified (7105)`
No version specified.

`rescode.err_cbf_obj_sense (7101)`
An invalid objective sense is specified.

`rescode.err_cbf_parse (7100)`
An error occurred while parsing an CBF file.

`rescode.err_cbf_syntax (7106)`
Invalid syntax.

`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_few_variables (7117)`
Too few variables defined.

`rescode.err_cbf_too_many_constraints (7103)`
Too many constraints specified.

`rescode.err_cbf_too_many_ints (7120)`
Too many ints are specified.

`rescode.err_cbf_too_many_variables (7104)`
Too many variables specified.

`rescode.err_cbf_unsupported (7122)`
Unsupported feature is present.

`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_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.

www.EngineeringBooksPdf.com

`rescode.err_cone_index (1300)`
An index of a non-existing cone has been 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_overlap_append (1307)`
The cone to be appended has one variable which is already member of another cone.

`rescode.err_cone_rep_var (1303)`
A variable is included multiple times in the cone.

`rescode.err_cone_size (1301)`
A cone with too few members is specified.

`rescode.err_cone_type (1305)`
Invalid cone type specified.

`rescode.err_cone_type_str (1306)`
Invalid cone type specified.

`rescode.err_data_file_ext (1055)`
The data file format cannot be determined from the file name.

`rescode.err_dup_name (1071)`
The same name was used multiple times for the same problem item type.

`rescode.err_duplicate_aij (1385)`
An element in the A matrix is specified twice.

`rescode.err_duplicate_barvariable_names (4502)`
Two barvariable names are identical.

`rescode.err_duplicate_cone_names (4503)`
Two cone names are identical.

`rescode.err_duplicate_constraint_names (4500)`
Two constraint names are identical.

`rescode.err_duplicate_variable_names (4501)`
Two variable names are identical.

`rescode.err_end_of_file (1059)`
End of file reached.

`rescode.err_factor (1650)`
An error occurred while factorizing a matrix.

`rescode.err_feasrepair_cannot_relax (1700)`
An optimization problem cannot be relaxed. This is the case e.g. for general nonlinear optimization problems.

`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_feasrepair_solving_relaxed (1701)`
The relaxed problem could not be solved to optimality. Please consult the log file for further details.

`rescode.err_file_license (1007)`
Invalid license file.

`rescode.err_file_open (1052)`
Error while opening a file.

www.EngineeringBooksPdf.com

`rescode.err_file_read (1053)`
File read error.

`rescode.err_file_write (1054)`
File write error.

`rescode.err_first (1261)`
Invalid `first`.

`rescode.err_firsti (1285)`
Invalid `firsti`.

`rescode.err_firstj (1287)`
Invalid `firstj`.

`rescode.err_fixed_bound_values (1425)`
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_flexlm (1014)`
The FLEXlm license manager reported an error.

`rescode.err_global_inv_conic_problem (1503)`
The global optimizer can only be applied to problems without semidefinite variables.

`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_huge_c (1375)`
A huge value in absolute size is specified for one c_j.

`rescode.err_identical_tasks (3101)`
Some tasks related to this function call were identical. Unique tasks were expected.

`rescode.err_in_argument (1200)`
A function argument is incorrect.

`rescode.err_index (1235)`
An index is out of range.

`rescode.err_index_arr_is_too_large (1222)`
An index in an array argument is too large.

`rescode.err_index_arr_is_too_small (1221)`
An index in an array argument is too small.

`rescode.err_index_is_too_large (1204)`
An index in an argument is too large.

`rescode.err_index_is_too_small (1203)`
An index in an argument is too small.

`rescode.err_inf_dou_index (1219)`
A double information index is out of range for the specified type.

`rescode.err_inf_dou_name (1230)`
A double information name is invalid.

`rescode.err_inf_int_index (1220)`
An integer information index is out of range for the specified type.

`rescode.err_inf_int_name (1231)`
An integer information name is invalid.

`rescode.err_inf_lint_index (1225)`
A long integer information index is out of range for the specified type.

www.EngineeringBooksPdf.com

`rescode.err_inf_lint_name (1234)`
A long integer information name is invalid.

`rescode.err_inf_type (1232)`
The information type is invalid.

`rescode.err_infeas_undefined (3910)`
The requested value is not defined for this solution type.

`rescode.err_infinite_bound (1400)`
A numerically huge bound value is specified.

`rescode.err_int64_to_int32_cast (3800)`
An 32 bit integer could not cast to a 64 bit integer.

`rescode.err_internal (3000)`
An internal error occurred. Please report this problem.

`rescode.err_internal_test_failed (3500)`
An internal unit test function failed.

`rescode.err_inv_aptre (1253)`
`aptre[j]` is strictly smaller than `aptrb[j]` for some `j`.

`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_cone_type (1272)`
Invalid cone type code is encountered.

`rescode.err_inv_cone_type_str (1271)`
Invalid cone type string encountered.

`rescode.err_inv_marki (2501)`
Invalid value in marki.

`rescode.err_inv_markj (2502)`
Invalid value in markj.

`rescode.err_inv_name_item (1280)`
An invalid name item code is used.

`rescode.err_inv_numi (2503)`
Invalid numi.

`rescode.err_inv_numj (2504)`
Invalid numj.

`rescode.err_inv_optimizer (1550)`
An invalid optimizer has been chosen for the problem. This means that the simplex or the conic optimizer is chosen to optimize a nonlinear problem.

`rescode.err_inv_problem (1500)`
Invalid problem type. Probably a nonconvex problem has been specified.

`rescode.err_inv_qcon_subi (1405)`
Invalid value in `qcsubi`.

`rescode.err_inv_qcon_subj (1406)`
Invalid value in `qcsubj`.

`rescode.err_inv_qcon_subk (1404)`
Invalid value in `qcsubk`.

www.EngineeringBooksPdf.com

`rescode.err_inv_qcon_val (1407)`
Invalid value in `qcval`.

`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_sk (1270)`
Invalid status key code.

`rescode.err_inv_sk_str (1269)`
Invalid status key string encountered.

`rescode.err_inv_skc (1267)`
Invalid value in `skc`.

`rescode.err_inv_skn (1274)`
Invalid value in `skn`.

`rescode.err_inv_skx (1268)`
Invalid value in `skx`.

`rescode.err_inv_var_type (1258)`
An invalid variable type is specified for a variable.

`rescode.err_invalid_accmode (2520)`
An invalid access mode is specified.

`rescode.err_invalid_aij (1473)`
$a_{i,j}$ contains an invalid floating point value, i.e. a `NaN` or an infinite value.

`rescode.err_invalid_ampl_stub (3700)`
Invalid AMPL stub.

`rescode.err_invalid_barvar_name (1079)`
An invalid symmetric matrix variable name is used.

`rescode.err_invalid_compression (1800)`
Invalid compression type.

`rescode.err_invalid_con_name (1076)`
An invalid constraint name is used.

`rescode.err_invalid_cone_name (1078)`
An invalid cone name is used.

`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_general_nl (4010)`
The file format does not support a problem with general nonlinear terms.

`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_name (1056)`
An invalid file name has been specified.

`rescode.err_invalid_format_type (1283)`
Invalid format type.

`rescode.err_invalid_idx (1246)`
A specified index is invalid.

www.EngineeringBooksPdf.com

`rescode.err_invalid_iomode (1801)`
Invalid io mode.

`rescode.err_invalid_max_num (1247)`
A specified index is invalid.

`rescode.err_invalid_name_in_sol_file (1170)`
An invalid name occurred in a solution file.

`rescode.err_invalid_obj_name (1075)`
An invalid objective name is specified.

`rescode.err_invalid_objective_sense (1445)`
An invalid objective sense is specified.

`rescode.err_invalid_problem_type (6000)`
An invalid problem type.

`rescode.err_invalid_sol_file_name (1057)`
An invalid file name has been specified.

`rescode.err_invalid_stream (1062)`
An invalid stream is referenced.

`rescode.err_invalid_surplus (1275)`
Invalid surplus.

`rescode.err_invalid_sym_mat_dim (3950)`
A sparse symmetric matrix of invalid dimension is specified.

`rescode.err_invalid_task (1064)`
The `task` is invalid.

`rescode.err_invalid_utf8 (2900)`
An invalid UTF8 string is encountered.

`rescode.err_invalid_var_name (1077)`
An invalid variable name is used.

`rescode.err_invalid_wchar (2901)`
An invalid `wchar` string is encountered.

`rescode.err_invalid_whichsol (1228)`
`whichsol` is invalid.

`rescode.err_json_data (1179)`
Inconsistent data in JSON Task file

`rescode.err_json_format (1178)`
Error in an JSON Task file

`rescode.err_json_missing_data (1180)`
Missing data section in JSON task file.

`rescode.err_json_number_overflow (1177)`
Invalid number entry - wrong type or value overflow.

`rescode.err_json_string (1176)`
Error in JSON string.

`rescode.err_json_syntax (1175)`
Syntax error in an JSON data

`rescode.err_last (1262)`
Invalid index `last`. A given index was out of expected range.

`rescode.err_lasti (1286)`
Invalid `lasti`.

www.EngineeringBooksPdf.com

`rescode.err_lastj (1288)`
Invalid `lastj`.

`rescode.err_lau_arg_k (7012)`
Invalid argument k.

`rescode.err_lau_arg_m (7010)`
Invalid argument m.

`rescode.err_lau_arg_n (7011)`
Invalid argument n.

`rescode.err_lau_arg_trans (7018)`
Invalid argument trans.

`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_invalid_lower_triangular_matrix (7002)`
An invalid lower triangular matrix.

`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_lau_not_positive_definite (7001)`
A matrix is not positive definite.

`rescode.err_lau_singular_matrix (7000)`
A matrix is singular.

`rescode.err_lau_unknown (7005)`
An unknown error.

`rescode.err_license (1000)`
Invalid license.

`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_expired (1001)`
The license has expired.

`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_license_invalid_hostid (1025)`
The host ID specified in the license file does not match the host ID of the computer.

`rescode.err_license_max (1016)`
Maximum number of licenses is reached.

`rescode.`
The MOSEKLM license manager daemon is not up and running.

`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.

www.EngineeringBooksPdf.com

`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_server (1015)`
The license server is not responding.

`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_version (1002)`
The license is valid for another version of **MOSEK**.

`rescode.err_link_file_dll (1040)`
A file cannot be linked to a stream in the DLL version.

`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_lower_bound_is_a_nan (1390)`
The lower bound specified is not a number (nan).

`rescode.err_lp_dup_slack_name (1152)`
The name of the slack variable added to a ranged constraint already exists.

`rescode.err_lp_empty (1151)`
The problem cannot be written to an LP formatted file.

`rescode.err_lp_file_format (1157)`
Syntax error in an LP file.

`rescode.err_lp_format (1160)`
Syntax error in an LP file.

`rescode.err_lp_free_constraint (1155)`
Free constraints cannot be written in LP file format.

`rescode.err_lp_incompatible (1150)`
The problem cannot be written to an LP formatted file.

`rescode.err_lp_invalid_con_name (1171)`
A constraint name is invalid when used in an LP formatted file.

`rescode.err_lp_invalid_var_name (1154)`
A variable name is invalid when used in an LP formatted file.

`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_lu_max_num_tries (2800)`
Could not compute the LU factors of the matrix within the maximum number of allowed tries.

`rescode.err_max_len_is_too_small (1289)`
An maximum length that is too small has been specified.

www.EngineeringBooksPdf.com

`rescode.err_maxnumbarvar (1242)`
The maximum number of semidefinite variables specified is smaller than the number of semidefinite variables in the task.

`rescode.err_maxnumcon (1240)`
The maximum number of constraints specified is smaller than the number of constraints in the task.

`rescode.err_maxnumcone (1304)`
The value specified for `maxnumcone` is too small.

`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_maxnumvar (1241)`
The maximum number of variables specified is smaller than the number of variables in the task.

`rescode.err_mio_internal (5010)`
A fatal error occurred in the mixed integer optimizer. Please contact **MOSEK** support.

`rescode.err_mio_invalid_node_optimizer (7131)`
An invalid node optimizer was selected for the problem type.

`rescode.err_mio_invalid_root_optimizer (7130)`
An invalid root optimizer was selected for the problem type.

`rescode.err_mio_no_optimizer (1551)`
No optimizer is available for the current class of integer optimization problems.

`rescode.err_mio_not_loaded (1553)`
The mixed-integer optimizer is not loaded.

`rescode.err_missing_license_file (1008)`
MOSEK cannot license file or a token server. See the **MOSEK** installation manual for details.

`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_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_cone_type (1117)`
Invalid cone type specified in a `CSECTION`.

`rescode.err_mps_duplicate_q_element (1121)`
Duplicate elements is specfied in a Q matrix.

`rescode.err_mps_file (1100)`
An error occurred while reading an MPS file.

`rescode.err_mps_inv_bound_key (1108)`
An invalid bound key 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_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_inv_sec_name (1109)`
An invalid section name occurred in an MPS file.

www.EngineeringBooksPdf.com

`rescode.err_mps_inv_sec_order (1115)`
The sections in the MPS data file are not in the correct order.

`rescode.err_mps_invalid_obj_name (1128)`
An invalid objective name is specified.

`rescode.err_mps_invalid_objsense (1122)`
An invalid objective sense is specified.

`rescode.err_mps_mul_con_name (1112)`
A constraint name was specified multiple times in the `ROWS` section.

`rescode.err_mps_mul_csec (1116)`
Multiple `CSECTION`s are given the same name.

`rescode.err_mps_mul_qobj (1114)`
The Q term in the objective is specified multiple times in the MPS data file.

`rescode.err_mps_mul_qsec (1113)`
Multiple `QSECTION`s are specified for a constraint in the MPS data file.

`rescode.err_mps_no_objective (1110)`
No objective is defined in an MPS file.

`rescode.err_mps_non_symmetric_q (1120)`
A non symmetric matrice has been speciefied.

`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_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_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_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_mul_a_element (1254)`
An element in A is defined multiple times.

`rescode.err_name_is_null (1760)`
The name buffer is a NULL pointer.

`rescode.err_name_max_len (1750)`
A name is longer than the buffer that is supposed to hold it.

`rescode.err_nan_in_blc (1461)`
l^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_buc (1462)`
u^c contains an invalid floating point value, i.e. a `NaN`.

www.EngineeringBooksPdf.com

`rescode.err_nan_in_bux (1472)`
u^x 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_double_data (1450)`
An invalid floating point value was used in some double data.

`rescode.err_negative_append (1264)`
Cannot append a negative number.

`rescode.err_negative_surplus (1263)`
Negative surplus.

`rescode.err_newer_dll (1036)`
The dynamic link library is newer than the specified version.

`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_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_basis_sol (1600)`
No basic solution is defined.

`rescode.err_no_dual_for_itg_sol (2950)`
No dual information is available for the integer solution.

`rescode.err_no_dual_infeas_cer (2001)`
A certificate of infeasibility is not available.

`rescode.err_no_init_env (1063)`
`env` is not initialized.

`rescode.err_no_optimizer_var_type (1552)`
No optimizer is available for this class of optimization problems.

`rescode.err_no_primal_infeas_cer (2000)`
A certificate of primal infeasibility is not available.

`rescode.err_no_snx_for_bas_sol (2953)`
s_n^x is not available for the basis solution.

`rescode.err_no_solution_in_callback (2500)`
The required solution is not available.

`rescode.err_non_unique_array (5000)`
An array does not contain unique elements.

`rescode.err_nonconvex (1291)`
The optimization problem is nonconvex.

`rescode.err_nonlinear_equality (1290)`
The model contains a nonlinear equality which defines a nonconvex set.

`rescode.err_nonlinear_functions_not_allowed (1428)`
An operation that is invalid for problems with nonlinear functions defined has been attempted.

`rescode.err_nonlinear_ranged (1292)`
Nonlinear constraints with finite lower and upper bound always define a nonconvex feasible set.

`rescode.err_nr_arguments (1199)`
Incorrect number of function arguments.

www.EngineeringBooksPdf.com

`rescode.err_null_env (1060)`
`env` is a NULL pointer.

`rescode.err_null_pointer (1065)`
An argument to a function is unexpectedly a NULL pointer.

`rescode.err_null_task (1061)`
`task` is a NULL pointer.

`rescode.err_numconlim (1250)`
Maximum number of constraints limit is exceeded.

`rescode.err_numvarlim (1251)`
Maximum number of variables limit is exceeded.

`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_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_objective_range (1260)`
Empty objective range.

`rescode.err_older_dll (1035)`
The dynamic link library is older than the specified version.

`rescode.err_open_dl (1030)`
A dynamic link library could not be opened.

`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_opf_premature_eof (1172)`
Premature end of file in an OPF file.

`rescode.err_optimizer_license (1013)`
The optimizer required is not licensed.

`rescode.err_overflow (1590)`
A computation produced an overflow i.e. a very large number.

`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_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.

www.EngineeringBooksPdf.com

`rescode.err_param_name_str (1208)`
The parameter name is not correct for a string parameter.

`rescode.err_param_type (1218)`
The parameter type is invalid.

`rescode.err_param_value_str (1217)`
The parameter value string is incorrect.

`rescode.err_platform_not_licensed (1019)`
A requested license feature is not available for the required platform.

`rescode.err_postsolve (1580)`
An error occurred during the postsolve. Please contact **MOSEK** support.

`rescode.err_pro_item (1281)`
An invalid problem is used.

`rescode.err_prob_license (1006)`
The software is not licensed to solve the problem.

`rescode.err_qcon_subi_too_large (1409)`
Invalid value in `qcsubi`.

`rescode.err_qcon_subi_too_small (1408)`
Invalid value in `qcsubi`.

`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_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_read_format (1090)`
The specified format cannot be read.

`rescode.err_read_lp_missing_end_tag (1159)`
Syntax error in LP file. Possibly missing End tag.

`rescode.err_read_lp_nonexisting_name (1162)`
A variable never occurred in objective or constraints.

`rescode.err_remove_cone_variable (1310)`
A variable cannot be removed because it will make a cone invalid.

`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_sen_bound_invalid_lo (3054)`
Analysis of lower bound requested for an index, where no lower bound exists.

`rescode.err_sen_bound_invalid_up (3053)`
Analysis of upper bound requested for an index, where no upper bound exists.

`rescode.err_sen_format (3050)`
Syntax error in sensitivity analysis file.

`rescode.err_sen_index_invalid (3055)`
Invalid range given in the sensitivity file.

`rescode.err_sen_index_range (3052)`
Index out of range in the sensitivity analysis file.

www.EngineeringBooksPdf.com

`rescode.err_sen_invalid_regexp (3056)`
Syntax error in regexp or regexp longer than 1024.

`rescode.err_sen_numerical (3058)`
Numerical difficulties encountered performing the sensitivity analysis.

`rescode.err_sen_solution_status (3057)`
No optimal solution found to the original problem given for sensitivity analysis.

`rescode.err_sen_undef_name (3051)`
An undefined name was encountered in the sensitivity analysis file.

`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_server_connect (8000)`
Failed to connect to remote solver server. The server string or the port string were invalid, or the server did not accept connection.

`rescode.err_server_protocol (8001)`
Unexpected message or data from solver server.

`rescode.err_server_status (8002)`
Server returned non-ok HTTP status code

`rescode.err_server_token (8003)`
The job ID specified is incorrect or invalid

`rescode.err_size_license (1005)`
The problem is bigger than the license.

`rescode.err_size_license_con (1010)`
The problem has too many constraints 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_size_license_numcores (3900)`
The computer contains more cpu cores than the license allows for.

`rescode.err_size_license_var (1011)`
The problem has too many variables to be solved with the available license.

`rescode.err_sol_file_invalid_number (1350)`
An invalid number is specified in a solution file.

`rescode.err_solitem (1237)`
The solution item number `solitem` is invalid. Please note that *solitem.snx* is invalid for the basic solution.

`rescode.err_solver_probtype (1259)`
Problem type does not match the chosen optimizer.

`rescode.err_space (1051)`
Out of space.

`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_sym_mat_duplicate (3944)`
A value in a symmetric matric as been specified more than once.

`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.

www.EngineeringBooksPdf.com

`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_invalid_col_index (3941)`
A column index specified for sparse symmetric matrix is invalid.

`rescode.err_sym_mat_invalid_row_index (3940)`
A row index specified for sparse symmetric matrix is invalid.

`rescode.err_sym_mat_invalid_value (3943)`
The numerical value specified in a sparse symmetric matrix is not a value floating value.

`rescode.err_sym_mat_not_lower_tringular (3942)`
Only the lower triangular part of sparse symmetric matrix should be specified.

`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_thread_cond_init (1049)`
Could not initialize a condition.

`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_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_toconic_constr_not_conic (7153)`
The constraint is not conic representable.

`rescode.err_toconic_constr_q_not_psd (7150)`
The matrix defining the quadratric part of constraint is not positive semidefinite.

`rescode.err_toconic_constraint_fx (7151)`
The quadratic constraint is an equality, thus not convex.

`rescode.err_toconic_constraint_ra (7152)`
The quadratic constraint has finite lower and upper bound, and therefore it is not convex.

`rescode.err_toconic_objective_not_psd (7155)`
The matrix defining the quadratric part of the objective function is not positive semidefinite.

`rescode.err_too_small_max_num_nz (1245)`
The maximum number of non-zeros specified is too small.

`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_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.

www.EngineeringBooksPdf.com

`rescode.err_undef_solution (1265)`
MOSEK has the following solution types:

- an interior-point solution,
- an basic solution,
- and an integer solution.

Each optimizer may set one or more of these solutions; e.g by default a successful optimization with the interior-point optimizer defines the interior-point solution, and, for linear problems, also the basic solution. This error occurs when asking for a solution or for information about a solution that is not defined.

`rescode.err_undefined_objective_sense (1446)`
The objective sense has not been specified before the optimization.

`rescode.err_unhandled_solution_status (6010)`
Unhandled solution status.

`rescode.err_unknown (1050)`
Unknown error.

`rescode.err_upper_bound_is_a_nan (1391)`
The upper bound specified is not a number (nan).

`rescode.err_upper_triangle (6020)`
An element in the upper triangle of a lower triangular matrix is specified.

`rescode.err_user_func_ret (1430)`
An user function reported an error.

`rescode.err_user_func_ret_data (1431)`
An user function returned invalid data.

`rescode.err_user_nlo_eval (1433)`
The user-defined nonlinear function reported an error.

`rescode.err_user_nlo_eval_hessubi (1440)`
The user-defined nonlinear function reported an invalid subscript in the Hessian.

`rescode.err_user_nlo_eval_hessubj (1441)`
The user-defined nonlinear function reported an invalid subscript in the Hessian.

`rescode.err_user_nlo_func (1432)`
The user-defined nonlinear function reported an error.

`rescode.err_whichitem_not_allowed (1238)`
whichitem is unacceptable.

`rescode.err_whichsol (1236)`
The solution defined by `whichsol` does not exists.

`rescode.err_write_lp_format (1158)`
Problem cannot be written as an LP file.

`rescode.err_write_lp_non_unique_name (1161)`
An auto-generated name is not unique.

`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_write_opf_invalid_var_name (1156)`
Empty variable names cannot be written to OPF files.

`rescode.err_writing_file (1166)`
An error occurred while writing file

www.EngineeringBooksPdf.com

`rescode.err_xml_invalid_problem_type (3600)`
The problem type is not supported by the XML format.

`rescode.err_y_is_undefined (1449)`
The solution item y is undefined.

16.5.3 Warning Codes

`rescode.wrn_ana_almost_int_bounds (904)`
This warning is issued by the problem analyzer if a constraint is bound nearly integral.

`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_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_empty_cols (902)`
This warning is issued by the problem analyzer, if columns, in which all coefficients are zero, are found.

`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_construct_invalid_sol_itg (807)`
The initial value for one or more of the integer variables is not feasible.

`rescode.wrn_construct_no_sol_itg (810)`
The construct solution requires an integer solution.

`rescode.wrn_construct_solution_infeas (805)`
After fixing the integer variables at the suggested values then the problem is infeasible.

`rescode.wrn_dropped_nz_qobj (201)`
One or more non-zero elements were dropped in the Q matrix in the objective.

`rescode.wrn_duplicate_barvariable_names (852)`
Two barvariable names are identical.

`rescode.wrn_duplicate_cone_names (853)`
Two cone names are identical.

`rescode.wrn_duplicate_constraint_names (850)`
Two constraint names are identical.

`rescode.wrn_duplicate_variable_names (851)`
Two variable names are identical.

`rescode.wrn_eliminator_space (801)`
The eliminator is skipped at least once due to lack of space.

`rescode.wrn_empty_name (502)`
A variable or constraint name is empty. The output file may be invalid.

`rescode.wrn_ignore_integer (250)`
Ignored integer constraints.

`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.

www.EngineeringBooksPdf.com

`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_large_bound (51)`
A numerically large bound value is specified.

`rescode.wrn_large_cj (57)`
A numerically large value is specified for one c_j.

`rescode.wrn_large_con_fx (54)`
An equality constraint is fixed to a numerically large value. This can cause numerical problems.

`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_license_expire (500)`
The license expires.

`rescode.wrn_license_feature_expire (505)`
The license expires.

`rescode.wrn_license_server (501)`
The license server is not responding.

`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_lp_old_quad_format (80)`
Missing '/2' after quadratic expressions in bound or objective.

`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_mps_split_bou_vector (72)`
A BOUNDS 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_rhs_vector (70)`
An RHS vector is split into several nonadjacent parts in an MPS file.

`rescode.wrn_name_max_len (65)`
A name is longer than the buffer that is supposed to hold it.

`rescode.wrn_no_dualizer (950)`
No automatic dualizer is available for the specified problem. The primal problem is solved.

`rescode.wrn_no_global_optimizer (251)`
No global optimizer is available.

`rescode.wrn_no_nonlinear_function_write (450)`
The problem contains a general nonlinear function in either the objective or the constraints. Such a nonlinear function cannot be written to a disk file. Note that quadratic terms when inputted explicitly can be written to disk.

`rescode.wrn_nz_in_upr_tri (200)`
Non-zero elements specified in the upper triangle of a matrix were ignored.

`rescode.wrn_open_param_file (50)`
The parameter file could not be opened.

www.EngineeringBooksPdf.com

`rescode.wrn_param_ignored_cmio (516)`
A parameter was ignored by the conic mixed integer optimizer.

`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_presolve_outofspace (802)`
The presolve is incomplete due to lack of space.

`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 problems, 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 problems, or to fix all the variables in the cone to 0.

`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_sol_filter (300)`
Invalid solution filter is specified.

`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_sym_mat_large (960)`
A numerically large value is specified for an $e_{i,j}$ element in E. The parameter *dparam.data_sym_mat_tol_large* controls when an $e_{i,j}$ is considered large.

`rescode.wrn_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_undef_sol_file_name (350)`
Undefined name occurred in a solution.

`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_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_zero_aij (63)`
One or more zero elements are specified in A.

www.EngineeringBooksPdf.com

`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_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.

16.6 Enumerations

`language`
Language selection constants

`language.eng`
English language selection

`language.dan`
Danish language selection

`accmode`
Constraint or variable access modes

`accmode.var`
Access data by columns (variable oriented)

`accmode.con`
Access data by rows (constraint oriented)

`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.

www.EngineeringBooksPdf.com

`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.

www.EngineeringBooksPdf.com

`simdupvec.free`
: The simplex optimizer can choose freely.

`simhotstart`
: Hot-start type employed by the simplex optimizer

`simhotstart.none`
: The simplex optimizer performs a coldstart.

`simhotstart.free`
: The simplex optimize chooses the hot-start type.

`simhotstart.status_keys`
: Only the status keys of the constraints and variables are used to choose the type of hot-start.

`intpnthotstart`
: Hot-start type employed by the interior-point optimizers.

`intpnthotstart.none`
: The interior-point optimizer performs a coldstart.

`intpnthotstart.primal`
: The interior-point optimizer exploits the primal solution only.

`intpnthotstart.dual`
: The interior-point optimizer exploits the dual solution only.

`intpnthotstart.primal_dual`
: The interior-point optimizer exploits both the primal and dual solution.

`callbackcode`
: Progress call-back codes

`callbackcode.begin_root_cutgen`
: The call-back function is called when root cut generation is started.

`callbackcode.im_root_cutgen`
: The call-back is called from within root cut generation at an intermediate stage.

`callbackcode.end_root_cutgen`
: The call-back function is called when root cut generation is is terminated.

`callbackcode.begin_optimizer`
: The call-back function is called when the optimizer is started.

`callbackcode.end_optimizer`
: The call-back function is called when the optimizer is terminated.

`callbackcode.begin_presolve`
: The call-back function is called when the presolve is started.

`callbackcode.update_presolve`
: The call-back function is called from within the presolve procedure.

`callbackcode.im_presolve`
: The call-back function is called from within the presolve procedure at an intermediate stage.

`callbackcode.end_presolve`
: The call-back function is called when the presolve is completed.

`callbackcode.begin_intpnt`
: The call-back function is called when the interior-point optimizer is started.

`callbackcode.intpnt`
: The call-back function is called from within the interior-point optimizer after the information database has been updated.

www.EngineeringBooksPdf.com

`callbackcode.im_intpnt`
: The call-back function is called at an intermediate stage within the interior-point optimizer where the information database has not been updated.

`callbackcode.end_intpnt`
: The call-back function is called when the interior-point optimizer is terminated.

`callbackcode.begin_conic`
: The call-back function is called when the conic optimizer is started.

`callbackcode.conic`
: The call-back function is called from within the conic optimizer after the information database has been updated.

`callbackcode.im_conic`
: The call-back function is called at an intermediate stage within the conic optimizer where the information database has not been updated.

`callbackcode.end_conic`
: The call-back function is called when the conic optimizer is terminated.

`callbackcode.primal_simplex`
: The call-back function is called from within the primal simplex optimizer.

`callbackcode.dual_simplex`
: The call-back function is called from within the dual simplex optimizer.

`callbackcode.begin_bi`
: The basis identification procedure has been started.

`callbackcode.im_bi`
: The call-back function is called from within the basis identification procedure at an intermediate point.

`callbackcode.end_bi`
: The call-back function is called when the basis identification procedure is terminated.

`callbackcode.begin_primal_bi`
: The call-back function is called from within the basis identification procedure when the primal phase is started.

`callbackcode.im_primal_bi`
: The call-back function is called from within the basis identification procedure at an intermediate point in the primal phase.

`callbackcode.update_primal_bi`
: The call-back function is called from within the basis identification procedure at an intermediate point in the primal phase.

`callbackcode.end_primal_bi`
: The call-back function is called from within the basis identification procedure when the primal phase is terminated.

`callbackcode.begin_dual_bi`
: The call-back function is called from within the basis identification procedure when the dual phase is started.

`callbackcode.im_dual_bi`
: The call-back function is called from within the basis identification procedure at an intermediate point in the dual phase.

`callbackcode.update_dual_bi`
: The call-back function is called from within the basis identification procedure at an intermediate point in the dual phase.

www.EngineeringBooksPdf.com

`callbackcode.end_dual_bi`
: The call-back function is called from within the basis identification procedure when the dual phase is terminated.

`callbackcode.begin_simplex_bi`
: The call-back function is called from within the basis identification procedure when the simplex clean-up phase is started.

`callbackcode.im_simplex_bi`
: The call-back function is called from within the basis identification procedure at an intermediate point in the simplex clean-up phase. The frequency of the call-backs is controlled by the *iparam.log_sim_freq* parameter.

`callbackcode.begin_primal_simplex_bi`
: The call-back function is called from within the basis identification procedure when the primal simplex clean-up phase is started.

`callbackcode.update_primal_simplex_bi`
: The call-back function is called from within the basis identification procedure at an intermediate point in the primal simplex clean-up phase. The frequency of the call-backs is controlled by the *iparam.log_sim_freq* parameter.

`callbackcode.end_primal_simplex_bi`
: The call-back function is called from within the basis identification procedure when the primal clean-up phase is terminated.

`callbackcode.begin_primal_dual_simplex_bi`
: The call-back function is called from within the basis identification procedure when the primal-dual simplex clean-up phase is started.

`callbackcode.update_primal_dual_simplex_bi`
: The call-back function is called from within the basis identification procedure at an intermediate point in the primal-dual simplex clean-up phase. The frequency of the call-backs is controlled by the *iparam.log_sim_freq* parameter.

`callbackcode.end_primal_dual_simplex_bi`
: The call-back function is called from within the basis identification procedure when the primal-dual clean-up phase is terminated.

`callbackcode.begin_dual_simplex_bi`
: The call-back function is called from within the basis identification procedure when the dual simplex clean-up phase is started.

`callbackcode.update_dual_simplex_bi`
: The call-back function is called from within the basis identification procedure at an intermediate point in the dual simplex clean-up phase. The frequency of the call-backs is controlled by the *iparam.log_sim_freq* parameter.

`callbackcode.end_dual_simplex_bi`
: The call-back function is called from within the basis identification procedure when the dual clean-up phase is terminated.

`callbackcode.end_simplex_bi`
: The call-back function is called from within the basis identification procedure when the simplex clean-up phase is terminated.

`callbackcode.begin_mio`
: The call-back function is called when the mixed-integer optimizer is started.

`callbackcode.im_mio`
: The call-back function is called at an intermediate point in the mixed-integer optimizer.

`callbackcode.new_int_mio`
: The call-back function is called after a new integer solution has been located by the mixed-integer optimizer.

www.EngineeringBooksPdf.com

`callbackcode.end_mio`
: The call-back function is called when the mixed-integer optimizer is terminated.

`callbackcode.begin_simplex`
: The call-back function is called when the simplex optimizer is started.

`callbackcode.begin_dual_simplex`
: The call-back function is called when the dual simplex optimizer started.

`callbackcode.im_dual_simplex`
: The call-back function is called at an intermediate point in the dual simplex optimizer.

`callbackcode.update_dual_simplex`
: The call-back function is called in the dual simplex optimizer.

`callbackcode.end_dual_simplex`
: The call-back function is called when the dual simplex optimizer is terminated.

`callbackcode.begin_primal_simplex`
: The call-back function is called when the primal simplex optimizer is started.

`callbackcode.im_primal_simplex`
: The call-back function is called at an intermediate point in the primal simplex optimizer.

`callbackcode.update_primal_simplex`
: The call-back function is called in the primal simplex optimizer.

`callbackcode.end_primal_simplex`
: The call-back function is called when the primal simplex optimizer is terminated.

`callbackcode.begin_primal_dual_simplex`
: The call-back function is called when the primal-dual simplex optimizer is started.

`callbackcode.im_primal_dual_simplex`
: The call-back function is called at an intermediate point in the primal-dual simplex optimizer.

`callbackcode.update_primal_dual_simplex`
: The call-back function is called in the primal-dual simplex optimizer.

`callbackcode.end_primal_dual_simplex`
: The call-back function is called when the primal-dual simplex optimizer is terminated.

`callbackcode.end_simplex`
: The call-back function is called when the simplex optimizer is terminated.

`callbackcode.begin_infeas_ana`
: The call-back function is called when the infeasibility analyzer is started.

`callbackcode.end_infeas_ana`
: The call-back function is called when the infeasibility analyzer is terminated.

`callbackcode.im_primal_sensivity`
: The call-back function is called at an intermediate stage of the primal sensitivity analysis.

`callbackcode.im_dual_sensivity`
: The call-back function is called at an intermediate stage of the dual sensitivity analysis.

`callbackcode.im_mio_intpnt`
: The call-back function is called at an intermediate point in the mixed-integer optimizer while running the interior-point optimizer.

`callbackcode.im_mio_primal_simplex`
: The call-back function is called at an intermediate point in the mixed-integer optimizer while running the primal simplex optimizer.

`callbackcode.im_mio_dual_simplex`
: The call-back function is called at an intermediate point in the mixed-integer optimizer while running the dual simplex optimizer.

www.EngineeringBooksPdf.com

`callbackcode.begin_primal_setup_bi`
The call-back function is called when the primal BI setup is started.

`callbackcode.end_primal_setup_bi`
The call-back function is called when the primal BI setup is terminated.

`callbackcode.begin_dual_setup_bi`
The call-back function is called when the dual BI phase is started.

`callbackcode.end_dual_setup_bi`
The call-back function is called when the dual BI phase is terminated.

`callbackcode.begin_primal_sensitivity`
Primal sensitivity analysis is started.

`callbackcode.end_primal_sensitivity`
Primal sensitivity analysis is terminated.

`callbackcode.begin_dual_sensitivity`
Dual sensitivity analysis is started.

`callbackcode.end_dual_sensitivity`
Dual sensitivity analysis is terminated.

`callbackcode.begin_license_wait`
Begin waiting for license.

`callbackcode.end_license_wait`
End waiting for license.

`callbackcode.im_license_wait`
MOSEK is waiting for a license.

`callbackcode.begin_qcqo_reformulate`
Begin QCQO reformulation.

`callbackcode.end_qcqo_reformulate`
End QCQO reformulation.

`callbackcode.im_qo_reformulate`
The call-back function is called at an intermediate stage of the conic quadratic reformulation.

`callbackcode.begin_to_conic`
Begin conic reformulation.

`callbackcode.end_to_conic`
End conic reformulation.

`callbackcode.begin_full_convexity_check`
Begin full convexity check.

`callbackcode.end_full_convexity_check`
End full convexity check.

`callbackcode.im_full_convexity_check`
The call-back function is called at an intermediate stage of the full convexity check.

`callbackcode.begin_primal_repair`
Begin primal feasibility repair.

`callbackcode.end_primal_repair`
End primal feasibility repair.

`callbackcode.begin_read`
MOSEK has started reading a problem file.

`callbackcode.im_read`
Intermediate stage in reading.

www.EngineeringBooksPdf.com

`callbackcode.end_read`
: **MOSEK** has finished reading a problem file.

`callbackcode.begin_write`
: **MOSEK** has started writing a problem file.

`callbackcode.end_write`
: **MOSEK** has finished writing a problem file.

`callbackcode.read_opf_section`
: A chunk of Q non-zeros has been read from a problem file.

`callbackcode.im_lu`
: The call-back function is called from within the LU factorization procedure at an intermediate point.

`callbackcode.im_order`
: The call-back function is called from within the matrix ordering procedure at an intermediate point.

`callbackcode.im_simplex`
: The call-back function is called from within the simplex optimizer at an intermediate point.

`callbackcode.read_opf`
: The call-back function is called from the OPF reader.

`callbackcode.write_opf`
: The call-back function is called from the OPF writer.

`callbackcode.solving_remote`
: The call-back function is called while the task is being solved on a remote server.

`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.

`conetype`
: Cone types

`conetype.quad`
: The cone is a quadratic cone.

`conetype.rquad`
: The cone is a rotated quadratic cone.

`nametype`
: Name types

www.EngineeringBooksPdf.com

`nametype.gen`
: General names. However, no duplicate and blank names are allowed.

`nametype.mps`
: MPS type names.

`nametype.lp`
: LP type names.

`symmattype`
: Cone types

`symmattype.sparse`
: Sparse symmetric matrix.

`dataformat`
: Data format types

`dataformat.extension`
: The file extension is used to determine the data file format.

`dataformat.mps`
: The data file is MPS formatted.

`dataformat.lp`
: The data file is LP formatted.

`dataformat.op`
: The data file is an optimization problem formatted file.

`dataformat.xml`
: The data file is an XML formatted file.

`dataformat.free_mps`
: The data a free MPS formatted file.

`dataformat.task`
: Generic task dump file.

`dataformat.cb`
: Conic benchmark format,

`dataformat.json_task`
: JSON based task format.

`dinfitem`
: Double information items

`dinfitem.bi_time`
: Time spent within the 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_dual_time`
: Time spent within the dual phase 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_clean_primal_time`
: Time spent within the primal clean-up optimizer of the basis identification procedure since its invocation.

`dinfitem.bi_clean_primal_dual_time`
: Time spent within the primal-dual clean-up optimizer of the basis identification procedure since its invocation.

www.EngineeringBooksPdf.com

`dinfitem.bi_clean_dual_time`
: Time spent within the dual clean-up optimizer of the basis identification procedure since its invocation.

`dinfitem.intpnt_time`
: Time spent within the interior-point optimizer since its invocation.

`dinfitem.intpnt_order_time`
: Order time (in seconds).

`dinfitem.intpnt_primal_obj`
: Primal objective value reported by the interior-point optimizer.

`dinfitem.intpnt_dual_obj`
: Dual objective value reported by the interior-point optimizer.

`dinfitem.intpnt_primal_feas`
: Primal feasibility measure reported by the interior-point optimizers. (For the interior-point optimizer this measure does not directly related to the original problem because a homogeneous model is employed).

`dinfitem.intpnt_dual_feas`
: Dual feasibility measure reported by the interior-point optimizer. (For the interior-point optimizer this measure does not directly related to the original problem because a homogeneous model is employed.)

`dinfitem.intpnt_opt_status`
: This measure should converge to $+1$ if the problem has a primal-dual optimal solution, and converge to -1 if 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.sim_time`
: Time spent in the simplex optimizer since invoking it.

`dinfitem.sim_primal_time`
: Time spent in the primal simplex optimizer since invoking it.

`dinfitem.sim_dual_time`
: Time spent in the dual simplex optimizer since invoking it.

`dinfitem.sim_primal_dual_time`
: Time spent in the primal-dual simplex optimizer since invoking it.

`dinfitem.sim_obj`
: Objective value reported by the simplex optimizer.

`dinfitem.sim_feas`
: Feasibility measure reported by the simplex optimizer.

`dinfitem.mio_time`
: Time spent in the mixed-integer optimizer.

`dinfitem.mio_root_presolve_time`
: Time spent in while presolving the root relaxation.

`dinfitem.mio_root_optimizer_time`
: Time spent in the optimizer while solving the root relaxation.

`dinfitem.mio_optimizer_time`
: Total time spent in the optimizer.

`dinfitem.mio_heuristic_time`
: Total time spent in the optimizer.

`dinfitem.to_conic_time`
: Time spent in the last to conic reformulation.

www.EngineeringBooksPdf.com

`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_obj_int`
The primal objective value corresponding to the best integer feasible solution. Please note that at least one integer feasible solution must have located i.e. check | *iinfitem.mio_num_int_solutions* |.

`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_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_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_user_obj_cut`
If the objective cut is used, then this information item has the value of the cut.

`dinfitem.mio_cmir_separation_time`
Seperation time for CMIR cuts.

`dinfitem.mio_clique_separation_time`
Seperation time for clique cuts.

`dinfitem.mio_knapsack_cover_separation_time`
Seperation time for knapsack cover.

`dinfitem.mio_gmi_separation_time`
Seperation time for GMI cuts.

`dinfitem.mio_implied_bound_time`
Seperation time for implied bound cuts.

`dinfitem.mio_root_cutgen_time`
Total time for cut generation.

`dinfitem.mio_probing_time`
Total time for probing.

`dinfitem.optimizer_time`
Total time spent in the optimizer since it was invoked.

`dinfitem.presolve_time`
Total time (in seconds) spent in the presolve since it was invoked.

`dinfitem.mio_dual_bound_after_presolve`
Value of the dual bound after presolve but before cut generation.

`dinfitem.presolve_eli_time`
Total time spent in the eliminator since the presolve was invoked.

www.EngineeringBooksPdf.com

`dinfitem.presolve_lindep_time`
Total time spent in the linear dependency checker since the presolve was invoked.

`dinfitem.rd_time`
Time spent reading the data file.

`dinfitem.sol_itr_primal_obj`
Primal objective value of the interior-point solution.

`dinfitem.sol_itr_pviolcon`
Maximal primal bound violation for x^c in the interior-point solution.

`dinfitem.sol_itr_pviolvar`
Maximal primal bound violation for x^x in the interior-point solution.

`dinfitem.sol_itr_pviolbarvar`
Maximal primal bound violation for $\overline{X}$ in the interior-point solution.

`dinfitem.sol_itr_pviolcones`
Maximal primal violation for primal conic constraints in the interior-point solution.

`dinfitem.sol_itr_dual_obj`
Dual objective value of the interior-point solution.

`dinfitem.sol_itr_dviolcon`
Maximal dual bound violation for x^c in the interior-point solution.

`dinfitem.sol_itr_dviolvar`
Maximal dual bound violation for x^x in the interior-point solution.

`dinfitem.sol_itr_dviolbarvar`
Maximal dual bound violation for $\overline{X}$ in the interior-point solution.

`dinfitem.sol_itr_dviolcones`
Maximal dual violation for dual conic constraints 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_barx`
Infinity norm of $\overline{X}$ in the interior-point solution.

`dinfitem.sol_itr_nrm_y`
Infinity norm of y 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_suc`
Infinity norm of s_u^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_sux`
Infinity norm of s_u^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_bars`
Infinity norm of $\overline{S}$ in the interior-point solution.

`dinfitem.sol_bas_primal_obj`
Primal objective value of the basic solution.

www.EngineeringBooksPdf.com

`dinfitem.sol_bas_pviolcon`
Maximal primal bound violation for x^c in the basic solution.

`dinfitem.sol_bas_pviolvar`
Maximal primal bound violation for x^x in the basic solution.

`dinfitem.sol_bas_dual_obj`
Dual objective value of the basic solution.

`dinfitem.sol_bas_dviolcon`
Maximal dual bound violation for x^c in the basic solution.

`dinfitem.sol_bas_dviolvar`
Maximal dual bound violation for x^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_barx`
Infinity norm of $\overline{X}$ in the basic solution.

`dinfitem.sol_bas_nrm_y`
Infinity norm of y in the basic solution.

`dinfitem.sol_bas_nrm_slc`
Infinity norm of s_l^c in the basic solution.

`dinfitem.sol_bas_nrm_suc`
Infinity norm of s_u^c in the basic solution.

`dinfitem.sol_bas_nrm_slx`
Infinity norm of s_l^x in the basic solution.

`dinfitem.sol_bas_nrm_sux`
Infinity norm of s_u^X in the basic solution.

`dinfitem.sol_itg_primal_obj`
Primal objective value of the integer solution.

`dinfitem.sol_itg_pviolcon`
Maximal primal bound violation for x^c in the integer solution.

`dinfitem.sol_itg_pviolvar`
Maximal primal bound violation for x^x in the integer solution.

`dinfitem.sol_itg_pviolbarvar`
Maximal primal bound violation for $\overline{X}$ in the integer solution.

`dinfitem.sol_itg_pviolcones`
Maximal primal violation for primal conic constraints in the integer solution.

`dinfitem.sol_itg_pviolitg`
Maximal violation for the integer constraints 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_nrm_barx`
Infinity norm of $\overline{X}$ in the integer solution.

`dinfitem.intpnt_factor_num_flops`
An estimate of the number of flops used in the factorization.

www.EngineeringBooksPdf.com

`dinfitem.qcqo_reformulate_time`
Time spent with conic quadratic reformulation.

`dinfitem.qcqo_reformulate_max_perturbation`
Maximum absolute diagonal perturbation occuring during the QCQO reformulation.

`dinfitem.qcqo_reformulate_worst_cholesky_diag_scaling`
Worst Cholesky diagonal scaling.

`dinfitem.qcqo_reformulate_worst_cholesky_column_scaling`
Worst Cholesky column scaling.

`dinfitem.primal_repair_penalty_obj`
The optimal objective value of the penalty function.

`feature`
License feature

`feature.pts`
Base system.

`feature.pton`
Nonlinear extension.

`liinfitem`
Long integer information items.

`liinfitem.mio_presolved_anz`
Number of non-zero entries in the constraint matrix of presolved problem.

`liinfitem.mio_simplex_iter`
Number of simplex iterations performed by the mixed-integer optimizer.

`liinfitem.mio_intpnt_iter`
Number of interior-point iterations performed by the mixed-integer optimizer.

`liinfitem.bi_primal_iter`
Number of primal pivots performed in the basis identification.

`liinfitem.bi_dual_iter`
Number of dual pivots performed in the basis identification.

`liinfitem.bi_clean_primal_iter`
Number of primal clean iterations performed in the basis identification.

`liinfitem.bi_clean_primal_dual_iter`
Number of primal-dual 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_dual_sub_iter`
Number of primal-dual subproblem clean iterations performed in the basis identification.

`liinfitem.bi_clean_primal_dual_deg_iter`
Number of primal-dual degenerate clean iterations performed in the basis identification.

`liinfitem.bi_clean_dual_deg_iter`
Number of dual degenerate clean iterations performed in the basis identification.

`liinfitem.intpnt_factor_num_nz`
Number of non-zeros in factorization.

`liinfitem.rd_numanz`
Number of non-zeros in A that is read.

www.EngineeringBooksPdf.com

`liinfitem.rd_numqnz`
: Number of Q non-zeros.

`liinfitem.mio_sim_maxiter_setbacks`
: Number of times the the simplex optimizer has hit the maximum iteration limit when re-optimizing.

`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_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_up`
: Number of constraints with an upper bound and an infinite lower 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_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_var`
: Number of variables in the problem.

  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_up`
: Number of variables with an upper bound and an infinite lower bound. This value is set by

  This value is set by *task.analyzeproblem*.

`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_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*.

www.EngineeringBooksPdf.com

`iinfitem.ana_pro_num_var_cont`
Number of continuous variables.

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_int`
Number of general integer variables.

This value is set by *task.analyzeproblem*.

`iinfitem.optimize_response`
The response code returned by optimize.

`iinfitem.intpnt_iter`
Number of interior-point iterations since invoking the interior-point optimizer.

`iinfitem.intpnt_factor_dim_dense`
Dimension of the dense sub system in factorization.

`iinfitem.intpnt_solve_dual`
Non-zero if the interior-point optimizer is solving the dual problem.

`iinfitem.mio_node_depth`
Depth of the last node solved.

`iinfitem.mio_numcon`
Number of constraints in the problem solved by the mixed-integer optimizer.

`iinfitem.mio_numvar`
Number of variables in the problem solved by the mixed-integer optimizer.

`iinfitem.mio_numint`
Number of integer variables in the problem solved be the mixed-integer optimizer.

`iinfitem.mio_presolved_numcont`
Number of continuous variables in the problem solved be the mixed-integer optimizer.

`iinfitem.mio_presolved_numbin`
Number of binary variables in the problem solved be the mixed-integer optimizer.

`iinfitem.mio_presolved_numcon`
Number of constraints in the presolved problem.

`iinfitem.mio_presolved_numvar`
Number of variables in the presolved problem.

`iinfitem.mio_presolved_numint`
Number of integer variables in the presolved problem.

`iinfitem.mio_clique_table_size`
Size of the clique table.

`iinfitem.mio_construct_solution`
If this item has the value 0, then **MOSEK** did not try to construct an initial integer feasible solution. If the item has a positive value, then **MOSEK** successfully constructed an initial integer feasible solution.

`iinfitem.mio_construct_num_roundings`
Number of values in the integer solution that is rounded to an integer value.

`iinfitem.mio_num_int_solutions`
Number of integer feasible solutions that has been found.

`iinfitem.mio_obj_bound_defined`
Non-zero if a valid objective bound has been found, otherwise zero.

www.EngineeringBooksPdf.com

`iinfitem.mio_num_active_nodes`
: Number of active branch bound nodes.

`iinfitem.mio_num_relax`
: Number of relaxations solved during the optimization.

`iinfitem.mio_num_branch`
: Number of branches performed during the optimization.

`iinfitem.mio_total_num_cuts`
: Total number of cuts generated by the mixed-integer optimizer.

`iinfitem.mio_num_cmir_cuts`
: Number of Complemented Mixed Integer Rounding (CMIR) cuts.

`iinfitem.mio_num_clique_cuts`
: Number of clique cuts.

`iinfitem.mio_num_implied_bound_cuts`
: Number of implied bound cuts.

`iinfitem.mio_num_knapsack_cover_cuts`
: Number of clique cuts.

`iinfitem.mio_num_gomory_cuts`
: Number of Gomory cuts.

`iinfitem.mio_num_repeated_presolve`
: Number of times presolve was repeated at root.

`iinfitem.mio_initial_solution`
: Is non-zero if an initial integer solution is specified.

`iinfitem.mio_user_obj_cut`
: If it is non-zero, then the objective cut is used.

`iinfitem.mio_relgap_satisfied`
: Non-zero if relative gap is within tolerances.

`iinfitem.mio_absgap_satisfied`
: Non-zero if absolute gap is within tolerances.

`iinfitem.mio_near_relgap_satisfied`
: Non-zero if relative gap is within relaxed tolerances.

`iinfitem.mio_near_absgap_satisfied`
: Non-zero if absolute gap is within relaxed tolerances.

`iinfitem.rd_protype`
: Problem type.

`iinfitem.rd_numcon`
: Number of constraints read.

`iinfitem.rd_numvar`
: Number of variables read.

`iinfitem.rd_numbarvar`
: Number of variables read.

`iinfitem.rd_numintvar`
: Number of integer-constrained variables read.

`iinfitem.rd_numq`
: Number of nonempty Q matrices read.

`iinfitem.sim_dual_deg_iter`
: The number of dual degenerate iterations.

www.EngineeringBooksPdf.com

`iinfitem.sim_dual_inf_iter`
: The number of iterations taken with dual infeasibility.

`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_primal_iter`
: Number of primal simplex iterations during the last optimization.

`iinfitem.sim_dual_iter`
: Number of dual simplex iterations during the last optimization.

`iinfitem.sim_primal_dual_iter`
: Number of primal dual simplex iterations during the last optimization.

`iinfitem.intpnt_num_threads`
: Number of threads that the interior-point optimizer is using.

`iinfitem.sim_primal_inf_iter`
: The number of iterations taken with primal infeasibility.

`iinfitem.sim_primal_dual_inf_iter`
: The number of master iterations with dual infeasibility taken by the primal dual simplex algorithm.

`iinfitem.sim_primal_deg_iter`
: The number of primal degenerate iterations.

`iinfitem.sim_primal_dual_deg_iter`
: The number of degenerate major iterations taken by the primal dual simplex algorithm.

`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_dual_hotstart`
: If 1 then the dual simplex algorithm is solving from an advanced basis.

`iinfitem.sim_primal_dual_hotstart`
: If 1 then the primal dual simplex algorithm is solving from an advanced basis.

`iinfitem.sim_primal_dual_hotstart_lu`
: If 1 then a valid basis factorization of full rank was located and used by the primal dual simplex algorithm.

`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.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.sim_numcon`
: Number of constraints in the problem solved by the simplex optimizer.

www.EngineeringBooksPdf.com

`iinfitem.sim_numvar`
: Number of variables in the problem solved by the simplex optimizer.

`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.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.

`iinfitem.rd_numcone`
: Number of conic constraints read.

`iinfitem.sim_solve_dual`
: Is non-zero if dual problem is solved.

`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 written.

`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.

www.EngineeringBooksPdf.com

`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.worst`
: The optimizer employs a worst bound node selection strategy.

`mionodeseltype.hybrid`
: The optimizer employs a hybrid 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.

www.EngineeringBooksPdf.com

`mpsformat.cplex`
The CPLEX compatible version of the MPS format is employed.

`msgkey`
Message keys

`msgkey.reading_file`

`msgkey.writing_file`

`msgkey.mps_selected`

`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.free`
The optimizer is chosen automatically.

`optimizertype.intpnt`
The interior-point optimizer is used.

`optimizertype.conic`
The optimizer for problems having conic constraints.

`optimizertype.primal_simplex`
The primal simplex optimizer is used.

`optimizertype.dual_simplex`
The dual simplex optimizer is used.

`optimizertype.free_simplex`
One of the simplex optimizers is used.

`optimizertype.mixed_int`
The mixed-integer optimizer.

`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.

www.EngineeringBooksPdf.com

`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.geco`
: General convex optimization.

`problemtype.conic`
: A conic optimization.

`problemtype.mixed`
: General nonlinear constraints and conic constraints. This combination can not be solved by **MOSEK**.

www.EngineeringBooksPdf.com

`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.near_prim_and_dual_feas`
  : The problem is at least nearly primal and dual feasible.

  `prosta.near_prim_feas`
  : The problem is at least nearly primal feasible.

  `prosta.near_dual_feas`
  : The problem is at least nearly 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.

www.EngineeringBooksPdf.com

`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.

`sensitivitytype.optimal_partition`
Optimal partition 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.

www.EngineeringBooksPdf.com

`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.near_optimal`
: The solution is nearly optimal.

`solsta.near_prim_feas`
: The solution is nearly primal feasible.

`solsta.near_dual_feas`
: The solution is nearly dual feasible.

`solsta.near_prim_and_dual_feas`
: The solution is nearly 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.near_prim_infeas_cer`
: The solution is almost a certificate of primal infeasibility.

`solsta.near_dual_infeas_cer`
: The solution is almost 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.

`solsta.near_integer_optimal`
: The primal solution is near integer optimal.

www.EngineeringBooksPdf.com

`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

www.EngineeringBooksPdf.com

`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.

16.7 Data Types

`env_t`
: The **MOSEK** Environment type.

`task_t`
: The **MOSEK** Task type.

`userhandle_t`
: A pointer to a generic user-defined structure.

`booleant`
: A signed integer interpreted as a boolean value.

`int32t`
: Signed 32bit integer.

`int64t`
: Signed 64bit integer.

`wchart`
: Wide char type. The actual type may differ depending on the platform; it is either a 16 or 32 bits signed or unsigned integer.

`realt`
: The floating point type used by **MOSEK**.

`string_t`
: The string type used by **MOSEK**. This is an UTF-8 encoded zero-terminated char string.

`iparam`
: int parameter type. See *Integer Parameters*

www.EngineeringBooksPdf.com

`dparam`
: double parameter type. See *Double Parameters*

`sparam`
: string parameter type. See *String Parameters*

`rescode`
: The return code type.

16.8 Class Env

`Env`
: The **MOSEK** global environment.

  `Members`

  env.__del__ – Free the underlying native allocation.

  env.axpy – Adds alpha times x to y.

  env.checkinall – Check in all unsued license features to the license token server.

  env.checkinlicense – Check in a license feature from the license server ahead of time.

  env.checkoutlicense – Check out a license feature from the license server ahead of time.

  env.computesparsecholesky – Computes a Cholesky factorization of sparse matrix.

  env.dot – Computes the inner product of two vectors.

  env.echointro – Prints an intro to message stream.

  env.gemm – Performs a dense matrix multiplication.

  env.gemv – Computes dense matrix times a dense vector product.

  env.getcodedesc – Obtains a short description of a response code.

  env.getversion – Obtains MOSEK version information.

  env.licensecleanup – Stops all threads and delete all handles used by the license system.

  env.linkfiletostream – Directs all output from a stream to a file.

  env.potrf – Computes a Cholesky factorization a dense matrix.

  env.putlicensecode – The purpose of this function is to input a runtime license code.

  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.

  env.set_Stream – Directs all output from a environment stream to a callback function.

  env.sparsetriangularsolvedense – Solves a sparse triangular system of linear equations.

  env.syeig – Computes all eigenvalues of a symmetric dense matrix.

  env.syevd – Computes all the eigenvalue and eigenvectors of a symmetric dense matrix, and thus its eigenvalue decomposition.

  env.syrk – Performs a rank-k update of a symmetric matrix.

`Env.Env()`
`Env.Env(licensefile)`
: Create a new environment.

`Parameters`

www.EngineeringBooksPdf.com

- `[in]` `licensefile` (`String`) – (Optional) License file to use.

`Env._del_()`
Free the underlying native allocation.

`Env.axpy(n, alpha, x, y)`
Adds αx to y.

`Parameters`

- `[in]` `n` (`int`) – Length of the vectors.
- `[in]` `alpha` (`double`) – The scalar that multiplies x.
- `[in]` `x` (`double[]`) – The x vector.
- `[io]` `y` (`double[]`) – The y vector.

`Env.checkinall()`
Check in all unsued license features to the license token server.

`Env.checkinlicense(feature)`
Check in a license feature to the license server. By default all licenses consumed by functions using a single environment is kept checked out for the lifetime of the **MOSEK** environment. This function checks in a given license feature to the license server immediately.

If the given license feature is not checked out or is in use by a call to *task.optimize* calling this function has no effect.

Please note that returning a license to the license server incurs a small overhead, so frequent calls to this function should be avoided.

`Parameters`

- `[in]` `feature` (*feature*) – Feature to check in to the license system.

`Env.checkoutlicense(feature)`
Check out a license feature from the license server. Normally the required license features will be automatically checked out the first time it is needed by the function *task.optimize*. This function can be used to check out one or more features ahead of time.

The license will remain checked out 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, only one feature will be checked out from the license server.

`Parameters`

- `[in]` `feature` (*feature*) – Feature to check out from the license system.

`perm, diag, lnzc, lptrc, lensubnval, lsubc, lvalc = Env.computesparsecholesky(multithread, ordermethod, tolsingular, anzc, aptrc, asubc, avalc)`
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.

To be precise then given a symmetric matrix $A \in \mathbb{R}^{n \times n}$ the function computes a nonsingular lower triangular matrix L and diagonal matrix D such that

$$LL^T - D = PAP^T$$

where P is a permutation matrix. If a pivot during the computation of the Cholesky factorization is less than

$$-\rho * \max((PAP^T)_{jj}, 1.0)$$

www.EngineeringBooksPdf.com

then the matrix is declared negative semidefinite. On the hand if a pivot is smaller than

$$\rho * \max((PAP^T)_{jj}, 1.0),$$

then D_{jj} is increased from zero to

$$\rho * \max((PAP^T)_{jj}, 1.0).$$

Therefore, if A is suffciently positive definite then D will be the zero matrix. ρ is set equal to value of tolsingular.

`Parameters`

- •`[in]` `multithread` (`int`) – If nonzero then the function may exploit multiple threads.
- •`[in]` `ordermethod` (`int`) – If nonzero, then a sparsity preservering ordering will be employed.
- •`[in]` `tolsingular` (`double`) – A positive parameter controling when a pivot is declared zero.
- •`[in]` `anzc` (`int[]`) – `anzc[j]` is the number of nonzeros in the jth column of `A`.
- •`[in]` `aptrc` (`long[]`) – `aptrc[j]` is a pointer to the first element in column `j`.
- •`[in]` `asubc` (`int[]`) – Row subscripts in each column stored in increasing order.
- •`[in]` `avalc` (`double[]`) – Values stores column wise.

`Return`

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

`xty` = `Env.dot(n, x, y)`

Computes the inner product of two vectors x, y of lenght $n \geq 0$, i.e

$$x \cdot y = \sum_{i=1}^{n} x_i y_i.$$

Note that if $n = 0$, then the results of the operation is 0.

`Parameters`

- •`[in]` `n` (`int`) – Length of the vectors.
- •`[in]` `x` (`double[]`) – The x vector.
- •`[in]` `y` (`double[]`) – The y vector.

`Return`

- •`xty` (`double`) – The result of the inner product between x and y.

`Env.echointro(longver)`

Prints an intro to message stream.

`Parameters`

- •`[in]` `longver` (`int`) – If non-zero, then the intro is slightly longer.

www.EngineeringBooksPdf.com

`Env.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 $op(X)$ return X if transX is YES, or X^T if set to NO. Dimensions of A, b must therefore match those of C.

The result of this operation is stored in C.

`Parameters`

- `[in]` `transa` (*transpose*) – Indicates whether the matrix A must be transposed.
- `[in]` `transb` (*transpose*) – Indicates whether the matrix B must be transposed.
- `[in]` `m` (`int`) – Indicates the number of rows of matrices A and C.
- `[in]` `n` (`int`) – Indicates the number of columns of matrices B and C.
- `[in]` `k` (`int`) – Specifies the number of columns of the matrix A and the number of rows of the matrix B.
- `[in]` `alpha` (`double`) – A scalar value multipling the result of the matrix multiplication.
- `[in]` `a` (`double[]`) – The pointer to the array storing matrix A in a column-major format.
- `[in]` `b` (`double[]`) – Indicates the number of rows of matrix B and columns of matrix A.
- `[in]` `beta` (`double`) – A scalar value that multiplies C.
- `[io]` `c` (`double[]`) – The pointer to the array storing matrix C in a column-major format.

`Env.gemv(transa, m, n, alpha, a, x, beta, y)`

Computes the multiplication of a scaled dense matrix times a dense vector product, plus a scaled dense vector. In formula

$$y = \alpha A x + \beta y,$$

or if trans is set to transpose.yes

$$y = \alpha A^T x + \beta y,$$

where α, β are scalar values. A is an $n \times m$ matrix, $x \in \mathbb{R}^m$ and $y \in \mathbb{R}^n$.

Note that the result is stored overwriting y.

`Parameters`

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

`symname, str = Env.getcodedesc(code)`

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

`Parameters`

- `[in]` `code` (*rescode*) – A valid **MOSEK** response code.

`Return`

www.EngineeringBooksPdf.com

- **symname** (str) – Symbolic name corresponding to code.
- **str** (str) – Obtains a short description of a response code.

major, minor, build, revision = Env.getversion()
Obtains **MOSEK** version information.
Return

- **major** (int) – Major version number.
- **minor** (int) – Minor version number.
- **build** (int) – Build number.
- **revision** (int) – Revision number.

Env.licensecleanup()
Stops all threads and delete 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.

Env.linkfiletostream(whichstream, filename, append)
Directs all output from a stream to a file.
Parameters

- **[in]** whichstream (*streamtype*) – Index of the stream.
- **[in]** filename (str) – Sends all output from the stream defined by whichstream to the file given by filename.
- **[in]** append (int) – If this argument is non-zero, the output is appended to the file.

Env.potrf(uplo, n, a)
Computes a Cholesky factorization of a real symmetric positive definite dense matrix.
Parameters

- **[in]** uplo (*uplo*) – Indicates whether the upper or lower triangular part of the matrix is stored.
- **[in]** n (int) – Dimension of the symmetric matrix.
- **[io]** a (double[]) – A symmetric matrix stored in column-major order. Only the lower or the upper triangular part is used, accordingly with the uplo parameter. It will contain the result on exit.

Env.putlicensecode(code)
The purpose of this function is to input a runtime license code.
Parameters

- **[in]** code (int[]) – A runtime license code.

Env.putlicensedebug(licdebug)
If licdebug is non-zero, then **MOSEK** will print debug info regarding the license checkout.
Parameters

- **[in]** licdebug (int) – If this argument is non-zero, then **MOSEK** will print debug info regarding the license checkout.

Env.putlicensepath(licensepath)
Set the path to the license file.
Parameters

- **[in]** licensepath (str) – A path specifycing where to search for the license.

Env.putlicensewait(licwait)
If licwait is non-zero, then **MOSEK** will wait for a license if no license is available. Moreover, licwait-1 is the number of milliseconds to wait between each check for an available license.
Parameters

www.EngineeringBooksPdf.com

- `[in]` `licwait` (`int`) – If this argument is non-zero, then **MOSEK** will wait for a license if no license is available. Moreover, `licwait-1` is the number of milliseconds to wait between each check for an available license.

`Env.set_Stream(whichstream, callback)`
Directs all output from a env stream to a callback function.
`Parameters`

- `[in]` `whichstream` (*streamtype*) – Index of the stream.
- `[in]` `callback` (`none`) – The callback function.

`Env.sparsetriangularsolvedense(transposed, lnzc, lptrc, lsubc, lvalc, b)`
The function can be used to 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`

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

`Env.syeig(uplo, n, a, w)`
Computes all eigenvalues of a real symmetric matrix A. Eigenvalues are stored in the w array.
`Parameters`

- `[in]` `uplo` (*uplo*) – Indicates whether the upper or lower triangular part is used.
- `[in]` `n` (`int`) – Dimension of the symmetric input matrix.
- `[in]` `a` (`double[]`) – A symmetric matrix stored in column-major order. Only the lower-triangular part is used.
- `[out]` `w` (`double[]`) – Array of minimum dimension n where eigenvalues will be stored.

`Env.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 the corresponding eigenvectors, stored in A as well.

Therefore, this function compute the eigenvalue decomposition of A as

$$A = UVU^T,$$

where $V = diag(w)$ and U contains the eigen-vectors of A.
`Parameters`

- `[in]` `uplo` (*uplo*) – Indicates whether the upper or lower triangular part is used.
- `[in]` `n` (`int`) – Dimension of symmetric input matrix.
- `[io]` `a` (`double[]`) – A symmetric matrix stored in column-major order. Only the lower-triangular part is used. It will be overwritten on exit.

www.EngineeringBooksPdf.com

- `[out] w` (`double[]`) – An array where eigenvalues will be stored. Its lenght must be at least the dimension of the input matrix.

`Env.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

$$C = \alpha A A^T + \beta C,$$

or

$$C = \alpha A^T A + \beta C.$$

In the first case $A \in \mathbb{R}^{k \times n}$, in the second $A \in \mathbb{R}^{n \times k}$.

Note that the results overwrite the matrix C.

`Parameters`

- `[in]` `uplo` (*uplo*) – Indicates whether the upper or lower triangular part of C is stored.
- `[in]` `trans` (*transpose*) – Indicates whether the matrix A must be transposed.
- `[in]` `n` (`int`) – Specifies the order of C.
- `[in]` `k` (`int`) – Indicates the number of rows or columns of A, and its rank.
- `[in]` `alpha` (`double`) – A scalar value multipling the result of the matrix multiplication.
- `[in]` `a` (`double[]`) – The pointer to the array storing matrix A in a column-major format.
- `[in]` `beta` (`double`) – A scalar value that multiplies C.
- `[io]` `c` (`double[]`) – The pointer to the array storing matrix C in a column-major format.

16.9 Class Task

`Task`

Represent an optimization task.

`Members`

task.analyzenames – Analyze the names and issue an error for the first invalid name.

task.analyzeproblem – Analyze the data of a task.

task.analyzesolution – Print information related to the quality of the solution.

task.appendbarvars – Appends a semidefinite variable of dimension dim to the problem.

task.appendcone – Appends a new cone constraint to the problem.

task.appendconeseq – Appends a new conic constraint to the problem.

task.appendconesseq – Appends a multiple conic constraints to the problem.

task.appendcons – Appends a number of constraints to the optimization task.

task.appendsparsesymmat – Appends a general sparse symmetric matrix to the vector E of symmetric matrixes.

task.appendvars – Appends a number of variables to the optimization task.

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.

www.EngineeringBooksPdf.com

task.asyncstop – Request that the job identified by the token is terminated.

task.basiscond – Computes conditioning information for the basis matrix.

task.checkconvexity – Checks if a quadratic optimization problem is convex.

task.checkmem – Checks the memory allocated by the task.

task.chgbound – Changes the bounds for one constraint or variable.

task.chgconbound – Changes the bounds for one constraint.

task.chgvarbound – Changes the bounds for one variable.

task.commitchanges – Commits all cached problem changes.

task.deletesolution – Undefine a solution and frees the memory it uses.

task.dualsensitivity – Performs sensitivity analysis on objective coefficients.

task.getacol – Obtains one column of the linear constraint matrix.

task.getacolnumnz – Obtains the number of non-zero elements in one column of the linear constraint matrix

task.getacolslicetrip – Obtains a sequence of columns from the coefficient matrix in triplet format.

task.getaij – Obtains a single coefficient in linear constraint matrix.

task.getapiecenumnz – Obtains the number non-zeros in a rectangular piece of the linear constraint matrix.

task.getarow – Obtains one row of the linear constraint matrix.

task.getarownumnz – Obtains the number of non-zero elements in one row of the linear constraint matrix

task.getarowslicetrip – Obtains a sequence of rows from the coefficient matrix in triplet format.

task.getaslice – Obtains a sequence of rows or columns from the coefficient matrix.

task.getaslicenumnz – Obtains the number of non-zeros in a slice of rows or columns of the coefficient matrix.

task.getbarablocktriplet – Obtains barA in block triplet form.

task.getbaraidx – Obtains information about an element barA.

task.getbaraidxij – Obtains information about an element barA.

task.getbaraidxinfo – Obtains the number terms in the weighted sum that forms a particular element in barA.

task.getbarasparsity – Obtains the sparsity pattern of the barA matrix.

task.getbarcblocktriplet – Obtains barc in block triplet form.

task.getbarcidx – Obtains information about an element in barc.

task.getbarcidxinfo – Obtains information about an element in barc.

task.getbarcidxj – Obtains the row index of an element in barc.

task.getbarcsparsity – Get the positions of the nonzero elements in barc.

task.getbarsj – Obtains the dual solution for a semidefinite variable.

task.getbarvarname – Obtains a name of a semidefinite variable.

task.getbarvarnameindex – Obtains the index of name of semidefinite variable.

task.getbarvarnamelen – Obtains the length of a name of a semidefinite variable.

task.getbarxj – Obtains the primal solution for a semidefinite variable.

www.EngineeringBooksPdf.com

task.getbound – Obtains bound information for one constraint or variable.

task.getboundslice – Obtains bounds information for a sequence of variables or constraints.

task.getc – Obtains all objective coefficients.

task.getcfix – Obtains the fixed term in the objective.

task.getcj – Obtains one coefficient of c.

task.getconbound – Obtains bound information for one constraint.

task.getconboundslice – Obtains bounds information for a slice of the constraints.

task.getcone – Obtains a conic constraint.

task.getconeinfo – Obtains information about a conic constraint.

task.getconename – Obtains a name of a cone.

task.getconenameindex – Checks whether the name somename has been assigned to any cone.

task.getconenamelen – Obtains the length of a name of a cone.

task.getconname – Obtains a name of a constraint.

task.getconnameindex – Checks whether the name somename has been assigned to any constraint.

task.getconnamelen – Obtains the length of a name of a constraint variable.

task.getcslice – Obtains a sequence of coefficients from the objective.

task.getdimbarvarj – Obtains the dimension of a symmetric matrix variable.

task.getdouinf – Obtains a double information item.

task.getdouparam – Obtains a double parameter.

task.getdualobj – Computes the dual objective value associated with the solution.

task.getdualsolutionnorms – Compute norms of the primal solution.

task.getdviolbarvar – Computes the violation of dual solution for a set of barx variables.

task.getdviolcon – Computes the violation of a dual solution associated with a set of constraints.

task.getdviolcones – Computes the violation of a solution for set of dual conic constraints.

task.getdviolvar – Computes the violation of a dual solution associated with a set of x variables.

task.getinfeasiblesubproblem – Obtains an infeasible sub problem.

task.getintinf – Obtains an integer information item.

task.getintparam – Obtains an integer parameter.

task.getlenbarvarj – Obtains the length if the j'th semidefinite variables.

task.getlintinf – Obtains an integer information item.

task.getmaxnumanz – Obtains number of preallocated non-zeros in the linear constraint matrix.

task.getmaxnumbarvar – Obtains maximum number of symmetric matrix variables that is reservered room for.

task.getmaxnumcon – Obtains the number of preallocated constraints in the optimization task.

task.getmaxnumcone – Obtains the number of preallocated cones in the optimization task.

task.getmaxnumqnz – Obtains the number of preallocated non-zeros for all quadratic terms in objective and constraints.

task.getmaxnumvar – Obtains the maximum number variables allowed.

task.getmemusage – Obtains information about the amount of memory used by a task.

www.EngineeringBooksPdf.com

task.getnumanz – Obtains the number of non-zeros in the coefficient matrix.

task.getnumanz64 – Obtains the number of non-zeros in the coefficient matrix.

task.getnumbarablocktriplets – Obtains an upper bound on the number of scalar elements in the block triplet form of bara.

task.getnumbaranz – Get the number of nonzero elements in barA.

task.getnumbarcblocktriplets – Obtains an upper bound on the number of elements in the block triplet form of barc.

task.getnumbarcnz – Obtains the number of nonzero elements in barc.

task.getnumbarvar – Obtains the number of semidefinite variables.

task.getnumcon – Obtains the number of constraints.

task.getnumcone – Obtains the number of cones.

task.getnumconemem – Obtains the number of members in a cone.

task.getnumintvar – Obtains the number of integer-constrained variables.

task.getnumparam – Obtains the number of parameters of a given type.

task.getnumqconknz – Obtains the number of non-zero quadratic terms in a constraint.

task.getnumqobjnz – Obtains the number of non-zero quadratic terms in the objective.

task.getnumsymmat – Get the number of symmetric matrixes stored.

task.getnumvar – Obtains the number of variables.

task.getobjname – Obtains the name assigned to the objective function.

task.getobjnamelen – Obtains the length of the name assigned to the objective function.

task.getobjsense – Gets the objective sense.

task.getprimalobj – Computes the primal objective value for the desired solution.

task.getprimalsolutionnorms – Compute norms of the primal solution.

task.getprobtype – Obtains the problem type.

task.getprosta – Obtains the problem status.

task.getpviolbarvar – Computes the violation of a primal solution for a list of barx variables.

task.getpviolcon – Computes the violation of a primal solution for a list of xc variables.

task.getpviolcones – Computes the violation of a solution for set of conic constraints.

task.getpviolvar – Computes the violation of a primal solution for a list of x variables.

task.getqconk – Obtains all the quadratic terms in a constraint.

task.getqobj – Obtains all the quadratic terms in the objective.

task.getqobjij – Obtains one coefficient from the quadratic term of the objective

task.getreducedcosts – Obtains the difference of (slx-sux) for a sequence of variables.

task.getskc – Obtains the status keys for the constraints.

task.getskcslice – Obtains the status keys for the constraints.

task.getskx – Obtains the status keys for the scalar variables.

task.getskxslice – Obtains the status keys for the variables.

task.getslc – Obtains the slc vector for a solution.

task.getslcslice – Obtains a slice of the slc vector for a solution.

task.getslx – Obtains the slx vector for a solution.

www.EngineeringBooksPdf.com

task.getslxslice – Obtains a slice of the slx vector for a solution.

task.getsnx – Obtains the snx vector for a solution.

task.getsnxslice – Obtains a slice of the snx vector for a solution.

task.getsolsta – Obtains the solution status.

task.getsolution – Obtains the complete solution.

task.getsolutioni – Obtains the solution for a single constraint or variable.

task.getsolutioninfo – Obtains information about of a solution.

task.getsolutionslice – Obtains a slice of the solution.

task.getsparsesymmat – Gets a single symmetric matrix from the matrix store.

task.getstrparam – Obtains the value of a string parameter.

task.getstrparamlen – Obtains the length of a string parameter.

task.getsuc – Obtains the suc vector for a solution.

task.getsucslice – Obtains a slice of the suc vector for a solution.

task.getsux – Obtains the sux vector for a solution.

task.getsuxslice – Obtains a slice of the sux vector for a solution.

task.getsymmatinfo – Obtains information of a matrix from the symmetric matrix storage E.

task.gettaskname – Obtains the task name.

task.gettasknamelen – Obtains the length the task name.

task.getvarbound – Obtains bound information for one variable.

task.getvarboundslice – Obtains bounds information for a slice of the variables.

task.getvarname – Obtains a name of a variable.

task.getvarnameindex – Checks whether the name somename has been assigned to any variable.

task.getvarnamelen – Obtains the length of a name of a variable variable.

task.getvartype – Gets the variable type of one variable.

task.getvartypelist – Obtains the variable type for one or more variables.

task.getxc – Obtains the xc vector for a solution.

task.getxcslice – Obtains a slice of the xc vector for a solution.

task.getxx – Obtains the xx vector for a solution.

task.getxxslice – Obtains a slice of the xx vector for a solution.

task.gety – Obtains the y vector for a solution.

task.getyslice – Obtains a slice of the y vector for a solution.

task.initbasissolve – Prepare a task for basis solver.

task.inputdata – Input the linear part of an optimization task in one function call.

task.isdouparname – Checks a double parameter name.

task.isintparname – Checks an integer parameter name.

task.isstrparname – Checks a string parameter name.

task.linkfiletostream – Directs all output from a task stream to a file.

task.onesolutionsummary – Prints a short summary for the specified solution.

task.optimize – Optimizes the problem.

www.EngineeringBooksPdf.com

task.optimizermt – Offload the optimization task to a solver server.

task.optimizersummary – Prints a short summary with optimizer statistics for last optimization.

task.primalrepair – The function repairs a primal infeasible optimization problem by adjusting the bounds on the constraints and variables.

task.primalsensitivity – Perform sensitivity analysis on bounds.

task.printdata – Prints a part of the problem data to a stream.

task.putacol – Replaces all elements in one column of A.

task.putacollist – Replaces all elements in several columns the linear constraint matrix by new values.

task.putacolslice – Replaces all elements in several columns the linear constraint matrix by new values.

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 A.

task.putarowlist – Replaces all elements in several rows the linear constraint matrix by new values.

task.putarowslice – Replaces all elements in several rows the linear constraint matrix by new values.

task.putbarablocktriplet – Inputs barA in block triplet form.

task.putbaraij – Inputs an element of barA.

task.putbarcblocktriplet – Inputs barC in block triplet form.

task.putbarcj – Changes one element in barc.

task.putbarsj – Sets the dual solution for a semidefinite variable.

task.putbarvarname – Puts the name of a semidefinite variable.

task.putbarxj – Sets the primal solution for a semidefinite variable.

task.putbound – Changes the bound for either one constraint or one variable.

task.putboundlist – Changes the bounds of constraints or variables.

task.putboundslice – Modifies bounds.

task.putcfix – Replaces the fixed term in the objective.

task.putcj – Modifies one linear coefficient in the objective.

task.putclist – Modifies a part of the linear objective coefficients.

task.putconbound – Changes the bound for one constraint.

task.putconboundlist – Changes the bounds of a list of constraints.

task.putconboundslice – Changes the bounds for a slice of the constraints.

task.putcone – Replaces a conic constraint.

task.putconename – Puts the name of a cone.

task.putconname – Puts the name of a constraint.

task.putcslice – Modifies a slice of the linear objective coefficients.

task.putdouparam – Sets a double parameter.

task.putintparam – Sets an integer parameter.

www.EngineeringBooksPdf.com

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

task.putmaxnumbarvar – Sets the number of preallocated symmetric matrix variables in the optimization task.

task.putmaxnumcon – Sets the number of preallocated constraints in the optimization task.

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

task.putmaxnumqnz – Changes the size of the preallocated storage for quadratic terms.

task.putmaxnumvar – Sets the number of preallocated variables in the optimization task.

task.putnadouparam – Sets a double parameter.

task.putnaintparam – Sets an integer parameter.

task.putnastrparam – Sets a string parameter.

task.putobjname – Assigns a new name to the objective.

task.putobjsense – Sets the objective sense.

task.putparam – Modifies the value of parameter.

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.

task.putskc – Sets the status keys for the constraints.

task.putskcslice – Sets the status keys for the constraints.

task.putskx – Sets the status keys for the scalar variables.

task.putskxslice – Sets the status keys for the variables.

task.putslc – Sets the slc vector for a solution.

task.putslcslice – Sets a slice of the slc vector for a solution.

task.putslx – Sets the slx vector for a solution.

task.putslxslice – Sets a slice of the slx vector for a solution.

task.putsnx – Sets the snx vector for a solution.

task.putsnxslice – Sets a slice of the snx vector for a solution.

task.putsolution – Inserts a solution.

task.putsolutioni – Sets the primal and dual solution information for a single constraint or variable.

task.putsolutionyi – Inputs the dual variable of a solution.

task.putstrparam – Sets a string parameter.

task.putsuc – Sets the suc vector for a solution.

task.putsucslice – Sets a slice of the suc vector for a solution.

task.putsux – Sets the sux vector for a solution.

task.putsuxslice – Sets a slice of the sux vector for a solution.

task.puttaskname – Assigns a new name to the task.

task.putvarbound – Changes the bound for one variable.

task.putvarboundlist – Changes the bounds of a list of variables.

www.EngineeringBooksPdf.com

task.putvarboundslice – Changes the bounds for a slice of the variables.

task.putvarname – Puts the name of a variable.

task.putvartype – Sets the variable type of one variable.

task.putvartypelist – Sets the variable type for one or more variables.

task.putxc – Sets the xc vector for a solution.

task.putxcslice – Sets a slice of the xc vector for a solution.

task.putxx – Sets the xx vector for a solution.

task.putxxslice – Obtains a slice of the xx vector for a solution.

task.puty – Sets the y vector for a solution.

task.putyslice – Sets a slice of the y vector for a solution.

task.readdata – Reads problem data from a file.

task.readdataformat – Reads problem data from a file.

task.readparamfile – Reads a parameter file.

task.readsolution – Reads a solution from a file.

task.readsummary – Prints information about last file read.

task.readtask – Load task data from a file.

task.removebarvars – The function removes a number of symmetric matrix.

task.removecones – Removes a conic constraint from the problem.

task.removecons – The function removes a number of constraints.

task.removevars – The function removes a number of variables.

task.resizetask – Resizes an optimization task.

task.sensitivityreport – Creates a sensitivity report.

task.set_Progress – Recieve callbacks about current status of the solver during optimization.

task.set_Stream – Directs all output from a task stream to a callback function.

task.setdefaults – Resets all parameters values.

task.solutiondef – Checks whether a solution is defined.

task.solutionsummary – Prints a short summary of the current solutions.

task.solvewithbasis – Solve a linear equation system involving a basis matrix.

task.strtoconetype – Obtains a cone type code.

task.strtosk – Obtains a status key.

task.toconic – Inplace reformulation of a QCQP to a COP

task.updatesolutioninfo – Update the information items related to the solution.

task.writeSC – Write problem to an SCopt file and a normal problem file.

task.writedata – Writes problem data to a file.

task.writejsonsol – Write a solution to a file.

task.writeparamfile – Writes all the parameters to a parameter file.

task.writesolution – Write a solution to a file.

task.writetask – Write a complete binary dump of the task data.

task.writetasksolverresult_file – Internal

www.EngineeringBooksPdf.com

`Task.Task(env)`
`Task.Task(other)`
Create a new Task in the given environment.
`Parameters`

- `[in]` `env` (*Env*) – Parent environment.
- `[in]` `other` (`Task`) – The task to copy.

`Task.analyzenames(whichstream, nametype)`
The function analyzes the names and issue an error if a name is invalid.
`Parameters`

- `[in]` `whichstream` (*streamtype*) – Index of the stream.
- `[in]` `nametype` (*nametype*) – The type of names e.g. valid in MPS or LP files.

`Task.analyzeproblem(whichstream)`
The function analyzes the data of task and writes out a report.
`Parameters`

- `[in]` `whichstream` (*streamtype*) – Index of the stream.

`Task.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`

- `[in]` `whichstream` (*streamtype*) – Index of the stream.
- `[in]` `whichsol` (*soltype*) – Selects a solution.

`Task.appendbarvars(dim)`
Appends a positive semidefinite matrix variable of dimension `dim` to the problem.
`Parameters`

- `[in]` `dim` (`int[]`) – Dimension of symmetric matrix variables to be added.

`Task.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`.

Depending on the value of `ct` this function appends a normal (*conetype.quad*) or rotated quadratic cone (*conetype.rquad*).

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:

www.EngineeringBooksPdf.com

•Quadratic cone (*conetype.quad*) :

$$\hat{x}_0 \geq \sqrt{\sum_{i=1}^{i<\text{nummem}} \hat{x}_i^2}$$

•Rotated quadratic cone (*conetype.rquad*) :

$$2\hat{x}_0\hat{x}_1 \geq \sum_{i=2}^{i<\text{nummem}} \hat{x}_i^2, \quad \hat{x}_0, \hat{x}_1 \geq 0$$

Please note that the sets of variables appearing in different conic constraints must be disjoint.

For an explained code example see Section *3.3*.

Parameters

- [in] ct (*conetype*) – Specifies the type of the cone.
- [in] conepar (double) – This argument is currently not used. It can be set to 0
- [in] submem (int[]) – Variable subscripts of the members in the cone.

Task.appendconeseq(ct, conepar, nummem, j)

Appends a new conic constraint to the problem. The function assumes the members of cone are sequential where the first member has index j and the last j+nummem-1.

Parameters

- [in] ct (*conetype*) – Specifies the type of the cone.
- [in] conepar (double) – This argument is currently not used. It can be set to 0
- [in] nummem (int) – Dimension of the conic constraint to be appended.
- [in] j (int) – Index of the first variable in the conic constraint.

Task.appendconesseq(ct, conepar, nummem, j)

Appends a number conic constraints to the problem. The kth cone is assumed to be of dimension nummem[k]. Moreover, is assumed that the first variable of the first cone has index j and the index of the variable in each cone are sequential. Finally, it assumed in the second cone is the last index of first cone plus one and so forth.

Parameters

- [in] ct (*conetype*) – Specifies the type of the cone.
- [in] conepar (double[]) – This argument is currently not used. It can be set to 0
- [in] nummem (int[]) – Number of member variables in the cone.
- [in] j (int) – Index of the first variable in the first cone to be appended.

Task.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

- [in] num (int) – Number of constraints which should be appended.

idx = Task.appendsparsesymmat(dim, subi, subj, valij)

MOSEK maintains a storage of symmetric data matrixes that is used to build the $\bar{c}$ and $\overline{A}$. The storage can be thought of as a vector of symmetric matrixes 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 matrixes. The vectors subi, subj, and valij contains the row subscripts, column subscripts

www.EngineeringBooksPdf.com

and values of each element in the symmetric matrix to be appended. Since the matrix that is appended is symmetric then 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

- [in] dim (int) – Dimension of the symmetric matrix that is appended.
- [in] subi (int[]) – Row subscript in the triplets.
- [in] subj (int[]) – Column subscripts in the triplets.
- [in] valij (double[]) – Values of each triplet.

Return

- idx (long) – Each matrix that is appended to E is assigned a unique index i.e. idx that can be used for later reference.

Task.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

- [in] num (int) – Number of variables which should be appended.

respavailable, resp, trm = Task.asyncgetresult(server, port, token)

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

Parameters

- [in] server (str) – Name or IP address of the solver server
- [in] port (str) – Network port of solver service
- [in] token (str) – The task token

Return

- respavailable (int) – Indicates if a remote response is available. If this is not true, res and trm should be ignored.
- resp (int) – Is either the response code from from the remote solver.
- trm (int) – Is either *rescodetype.ok* or a termination response code.

token = Task.asyncoptimize(server, port)

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

- [in] server (str) – Name or IP address of the solver server
- [in] port (str) – Network port of solver service

Return

- token (str) – Returns the task token

respavailable, resp, trm = Task.asyncpoll(server, port, token)

Requests information about the status of the remote job.

Parameters

- [in] server (str) – Name or IP address of the solver server
- [in] port (str) – Network port of solver service

www.EngineeringBooksPdf.com

- **[in]** `token` (`str`) – The task token

Return

- `respavailable` (`int`) – Indicates if a remote response is available. If this is not true, `res` and `trm` should be ignored.
- `resp` (`int`) – Is either the response code from from the remote solver.
- `trm` (`int`) – Is either *rescodetype.ok* or a termination response code.

`Task.asyncstop(server, port, token)`

Request that the job identified by the `token` is terminated.

Parameters

- **[in]** `server` (`str`) – Name or IP address of the solver server
- **[in]** `port` (`str`) – Network port of solver service
- **[in]** `token` (`str`) – The task token

`nrmbasis, nrminvbasis = Task.basiscond()`

If a basic solution is available and it defines a nonsingular basis, then this function computes the 1-norm estimate of the basis matrix and an 1-norm estimate for the inverse of the basis matrix. The 1-norm estimates are computed using the method outlined in *[Ste98]*, pp. 388-391.

By definition the 1-norm condition number of a matrix B is defined as

$$\mathcal{K}_1(B) := \|B\|_1 \|B^{-1}|.$$

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` (`double`) – An estimate for the 1 norm of the basis.
- `nrminvbasis` (`double`) – An estimate for the 1 norm of the inverse of the basis.

`Task.checkconvexity()`

This function checks if a quadratic optimization problem is convex. The amount of checking is controlled by *iparam.check_convexity*.

The function reports an error if the problem is not convex.

`Task.checkmem(file, line)`

Checks the memory allocated by the task.

Parameters

- **[in]** `file` (`str`) – File from which the function is called.
- **[in]** `line` (`int`) – Line in the file from which the function is called.

`Task.chgbound(accmode, i, lower, finite, value)`

Changes a bound for one constraint or variable. If `accmode` equals *accmode.con*, a constraint bound is changed, otherwise a variable bound is changed.

If `lower` is non-zero, then the lower bound is changed as follows:

$$\text{new lower bound} = \begin{cases} -\infty, & \texttt{finite} = 0, \\ \texttt{value} & \text{otherwise.} \end{cases}$$

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 bound, in particular, if the lower and upper bounds are identical, the bound key is changed to `fixed`.

Parameters

www.EngineeringBooksPdf.com

- [in] `accmode` (*accmode*) – Defines if operations are performed row-wise (constraint-oriented) or column-wise (variable-oriented).
- [in] `i` (`int`) – Index of the constraint or variable for which the bounds should be changed.
- [in] `lower` (`int`) – If non-zero, then the lower bound is changed, otherwise the upper bound is changed.
- [in] `finite` (`int`) – If non-zero, then `value` is assumed to be finite.
- [in] `value` (`double`) – New value for the bound.

`Task.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 bound, in particular, if the lower and upper bounds are identical, the bound key is changed to `fixed`.

`Parameters`

- [in] `i` (`int`) – Index of the constraint for which the bounds should be changed.
- [in] `lower` (`int`) – If non-zero, then the lower bound is changed, otherwise the upper bound is changed.
- [in] `finite` (`int`) – If non-zero, then `value` is assumed to be finite.
- [in] `value` (`double`) – New value for the bound.

`Task.chgvarbound(j, lower, finite, value)`

Changes a bound for on 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 bound, in particular, if the lower and upper bounds are identical, the bound key is changed to `fixed`.

`Parameters`

- [in] `j` (`int`) – Index of the variable for which the bounds should be changed.
- [in] `lower` (`int`) – If non-zero, then the lower bound is changed, otherwise the upper bound is changed.
- [in] `finite` (`int`) – If non-zero, then `value` is assumed to be finite.
- [in] `value` (`double`) – New value for the bound.

`Task.commitchanges()`

Commits all cached problem changes to the task. It is usually not necessary explicitly to call this function since changes will be committed automatically when required.

`Task.deletesolution(whichsol)`

Undefine a solution and frees the memory it uses.

www.EngineeringBooksPdf.com

Parameters

- [in] whichsol (*soltype*) – Selects a solution.

Task.dualsensitivity(subj, leftpricej, rightpricej, leftrangej, rightrangej)
Calculates sensitivity information for objective coefficients. The indexes of the coefficients to analyze are

$$\{\texttt{subj}[i] | i \in 0, \ldots, \texttt{numj} - 1\}$$

The results are returned so that e.g leftprice$[j]$ is the left shadow price of the objective coefficient with index subj$[j]$.

The type of sensitivity analysis to perform (basis or optimal partition) is controlled by the parameter *iparam.sensitivity_type*.

For an example, please see Section *15.1.4*.

Parameters

- [in] subj (int[]) – Index of objective coefficients to analyze.
- [out] leftpricej (double[]) – leftpricej$[j]$ is the left shadow price for the coefficients with index subj[j].
- [out] rightpricej (double[]) – rightpricej$[j]$ is the right shadow price for the coefficients with index subj[j].
- [out] leftrangej (double[]) – leftrangej$[j]$ is the left range β_1 for the coefficient with index subj[j].
- [out] rightrangej (double[]) – rightrangej$[j]$ is the right range β_2 for the coefficient with index subj[j].

nzj = Task.getacol(j, subj, valj)
Obtains one column of A in a sparse format.

Parameters

- [in] j (int) – Index of the column.
- [out] subj (int[]) – Index of the non-zeros in the row obtained.
- [out] valj (double[]) – Numerical values of the column obtained.

Return

- nzj (int) – Number of non-zeros in the column obtained.

nzj = Task.getacolnumnz(i)
Obtains the number of non-zero elements in one column of A.

Parameters

- [in] i (int) – Index of the column.

Return

- nzj (int) – Number of non-zeros in the jth row or column of A.

Task.getacolslicetrip(first, last, subi, subj, val)
Obtains a sequence of columns from A in a sparse triplet format.

Parameters

- [in] first (int) – Index of the first column in the sequence.
- [in] last (int) – Index of the last column in the sequence **plus one**.
- [out] subi (int[]) – Constraint subscripts.
- [out] subj (int[]) – Column subscripts.
- [out] val (double[]) – Values.

aij = Task.getaij(i, j)
Obtains a single coefficient in A.

www.EngineeringBooksPdf.com

Parameters

- [in] i (int) – Row index of the coefficient to be returned.
- [in] j (int) – Column index of the coefficient to be returned.

Return

- aij (double) – The required coefficient $a_{i,j}$.

numnz = Task.getapiecenumnz(firsti, lasti, firstj, lastj)

Obtains the number non-zeros in a rectangular piece of A, i.e. the number

$$|(i,j): \ a_{i,j} \neq 0, \ \texttt{firsti} \leq i \leq \texttt{lasti} - 1, \ \texttt{firstj} \leq j \leq \texttt{lastj} - 1\}|$$

where $|\mathcal{I}|$ means the number of elements in the set $\mathcal{I}$.

This function is not an efficient way to obtain the number of non-zeros in one row or column. In that case use the function *task.getarownumnz* or *task.getacolnumnz*.

Parameters

- [in] firsti (int) – Index of the first row in the rectangular piece.
- [in] lasti (int) – Index of the last row plus one in the rectangular piece.
- [in] firstj (int) – Index of the first column in the rectangular piece.
- [in] lastj (int) – Index of the last column plus one in the rectangular piece.

Return

- numnz (int) – Number of non-zero A elements in the rectangular piece.

nzi = Task.getarow(i, subi, vali)

Obtains one row of A in a sparse format.

Parameters

- [in] i (int) – Index of the row or column.
- [out] subi (int[]) – Index of the non-zeros in the row obtained.
- [out] vali (double[]) – Numerical values of the row obtained.

Return

- nzi (int) – Number of non-zeros in the row obtained.

nzi = Task.getarownumnz(i)

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

Parameters

- [in] i (int) – Index of the row or column.

Return

- nzi (int) – Number of non-zeros in the ith row of A.

Task.getarowslicetrip(first, last, subi, subj, val)

Obtains a sequence of rows from A in a sparse triplets format.

Parameters

- [in] first (int) – Index of the first row or column in the sequence.
- [in] last (int) – Index of the last row or column in the sequence **plus one**.
- [out] subi (int[]) – Constraint subscripts.
- [out] subj (int[]) – Column subscripts.
- [out] val (double[]) – Values.

Task.getaslice(accmode, first, last, ptrb, ptre, sub, val)

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

Parameters

www.EngineeringBooksPdf.com

- [in] accmode (*accmode*) – Defines whether a column slice or a row slice is requested.
- [in] first (int) – Index of the first row or column in the sequence.
- [in] last (int) – Index of the last row or column in the sequence **plus one**.
- [out] ptrb (long[]) – ptrb[t] is an index pointing to the first element in the tth row or column obtained.
- [out] ptre (long[]) – ptre[t] is an index pointing to the last element plus one in the tth row or column obtained.
- [out] sub (int[]) – Contains the row or column subscripts.
- [out] val (double[]) – Contains the coefficient values.

numnz = Task.getaslicenumnz(accmode, first, last)

Obtains the number of non-zeros in a slice of rows or columns of A.

Parameters

- [in] accmode (*accmode*) – Defines whether non-zeros are counted in a column slice or a row slice.
- [in] first (int) – Index of the first row or column in the sequence.
- [in] last (int) – Index of the last row or column **plus one** in the sequence.

Return

- numnz (long) – Number of non-zeros in the slice.

num = Task.getbarablocktriplet(subi, subj, subk, subl, valijkl)

Obtains $\overline{A}$ in block triplet form.

Parameters

- [out] subi (int[]) – Constraint index.
- [out] subj (int[]) – Symmetric matrix variable index.
- [out] subk (int[]) – Block row index.
- [out] subl (int[]) – Block column index.
- [out] valijkl (double[]) – A list indexes of the elements from symmetric matrix storage that appears in the weighted sum.

Return

- num (long) – Number of elements in the block triplet form.

i, j, num = Task.getbaraidx(idx, sub, weights)

Obtains information about an element in $\overline{A}$. Since $\overline{A}$ is a sparse matrix of symmetric matrixes then only the nonzero elements in $\overline{A}$ are stored in order to save space. Now $\overline{A}$ is stored vectorized form 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

- [in] idx (long) – Position of the element in the vectorized form.
- [out] sub (long[]) – A list indexes of the elements from symmetric matrix storage that appears in the weighted sum.
- [out] weights (double[]) – The weights associated with each term in the weighted sum.

Return

- i (int) – Row index of the element at position idx.
- j (int) – Column index of the element at position idx.
- num (long) – Number of terms in weighted sum that forms the element.

www.EngineeringBooksPdf.com

i, j = Task.getbaraidxij(idx)
Obtains information about an element in $\overline{A}$. Since $\overline{A}$ is a sparse matrix of symmetric matrixes only the nonzero elements in $\overline{A}$ are stored in order to save space. Now $\overline{A}$ is stored vectorized form 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

- [in] idx (long) – Position of the element in the vectorized form.

Return

- i (int) – Row index of the element at position idx.
- j (int) – Column index of the element at position idx.

num = Task.getbaraidxinfo(idx)
Each nonzero element in $\overline{A}_{ij}$ is formed as a weighted sum of symmetric matrices. Using this function the number terms in the weighted sum can be obtained. See description of *task.appendsparsesymmat* for details about the weighted sum.

Parameters

- [in] idx (long) – The internal position of the element that should be obtained information for.

Return

- num (long) – Number of terms in the weighted sum that forms the specified element in $\overline{A}$.

numnz = Task.getbarasparsity(idxij)
The matrix $\overline{A}$ is assumed to be a sparse matrix of symmetric matrices. This implies that many of elements in $\overline{A}$ is likely to be zero matrixes. 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 $\overline{A}$.

Parameters

- [out] idxij (long[]) – Position of each nonzero element in the vectorized form of $\overline{A}_{ij}$. Hence, idxij[k] is the vector position of the element in row subi[k] and column subj[k] of $\overline{A}_{ij}$.

Return

- numnz (long) – Number of nonzero elements in $\overline{A}$.

num = Task.getbarcblocktriplet(subj, subk, subl, valijkl)
Obtains $\overline{C}$ in block triplet form.

Parameters

- [out] subj (int[]) – Symmetric matrix variable index.
- [out] subk (int[]) – Block row index.
- [out] subl (int[]) – Block column index.
- [out] valijkl (double[]) – A list indexes of the elements from symmetric matrix storage that appears in the weighted sum.

Return

- num (long) – Number of elements in the block triplet form.

j, num = Task.getbarcidx(idx, sub, weights)
Obtains information about an element in $\bar{c}$.

Parameters

- [in] idx (long) – Index of the element that should be obtained information about.
- [out] sub (long[]) – Elements appearing the weighted sum.
- [out] weights (double[]) – Weights of terms in the weighted sum.

www.EngineeringBooksPdf.com

Return

- j (int) – Row index in $\bar{c}$.
- num (long) – Number of terms in the weighted sum.

num = Task.getbarcidxinfo(idx)

Obtains information about the $\bar{c}_{ij}$.

Parameters

- [in] idx (long) – Index of element that should be obtained information about. The value is an index of a symmetric sparse variable.

Return

- num (long) – Number of terms that appears in weighted that forms the requested element.

j = Task.getbarcidxj(idx)

Obtains the row index of an element in $\bar{c}$.

Parameters

- [in] idx (long) – Index of the element that should be obtained information about.

Return

- j (int) – Row index in $\bar{c}$.

numnz = Task.getbarcsparsity(idxj)

Internally only the nonzero elements of $\bar{c}$ is stored

in a vector. This function returns which elements $\bar{c}$ that are nonzero (in subj) and their internal position (in idx). Using the position detailed information about each nonzero $\overline{C}_j$ can be obtained using *task.getbarcidxinfo* and *task.getbarcidx*.

Parameters

- [out] idxj (long[]) – Internal positions of the nonzeros elements in $\bar{c}$.

Return

- numnz (long) – Number of nonzero elements in $\overline{C}$.

Task.getbarsj(whichsol, j, barsj)

Obtains the dual solution for a semidefinite variable. Only the lower triangle part of $\bar{s}_j$ is returned because the matrix by construction is symmetric. The format is that the columns are stored sequentially in the natural order.

Parameters

- [in] whichsol (*soltype*) – Selects a solution.
- [in] j (int) – Index of the semidefinite variable.
- [out] barsj (double[]) – Value of $\bar{s}_j$.

name = Task.getbarvarname(i)

Obtains a name of a semidefinite variable.

Parameters

- [in] i (int) – Index.

Return

- name (str) – The requested name is copied to this buffer.

asgn, index = Task.getbarvarnameindex(somename)

Obtains the index of name of semidefinite variable.

Parameters

- [in] somename (str) – The requested name is copied to this buffer.

Return

- asgn (int) – Is non-zero if the name somename is assigned to a semidefinite variable.
- index (int) – If the name somename is assigned to a semidefinite variable, then index is the name of the constraint.

www.EngineeringBooksPdf.com

`len` = `Task.getbarvarnamelen(i)`
Obtains the length of a name of a semidefinite variable.
Parameters

- `[in]` `i` (`int`) – Index.

Return

- `len` (`int`) – Returns the length of the indicated name.

`Task.getbarxj(whichsol, j, barxj)`
Obtains the primal solution for a semidefinite variable. Only the lower triangle part of $\bar{x}_j$ is returned because the matrix by construction is symmetric. The format is that the columns are stored sequentially in the natural order.
Parameters

- `[in]` `whichsol` (*soltype*) – Selects a solution.
- `[in]` `j` (`int`) – Index of the semidefinite variable.
- `[out]` `barxj` (`double[]`) – Value of $\overline{X}_j$.

`bk`, `bl`, `bu` = `Task.getbound(accmode, i)`
Obtains bound information for one constraint or variable.
Parameters

- `[in]` `accmode` (*accmode*) – Defines if operations are performed row-wise (constraint-oriented) or column-wise (variable-oriented).
- `[in]` `i` (`int`) – Index of the constraint or variable for which the bound information should be obtained.

Return

- `bk` (`int`) – Bound keys.
- `bl` (`double`) – Values for lower bounds.
- `bu` (`double`) – Values for upper bounds.

`Task.getboundslice(accmode, first, last, bk, bl, bu)`
Obtains bounds information for a sequence of variables or constraints.
Parameters

- `[in]` `accmode` (*accmode*) – Defines if operations are performed row-wise (constraint-oriented) or column-wise (variable-oriented).
- `[in]` `first` (`int`) – First index in the sequence.
- `[in]` `last` (`int`) – Last index plus 1 in the sequence.
- `[out]` `bk` (*boundkey*) – Bound keys.
- `[out]` `bl` (`double[]`) – Values for lower bounds.
- `[out]` `bu` (`double[]`) – Values for upper bounds.

`Task.getc(c)`
Obtains all objective coefficients c.
Parameters

- `[out]` `c` (`double[]`) – Linear terms of the objective as a dense vector. The lengths is the number of variables.

`cfix` = `Task.getcfix()`
Obtains the fixed term in the objective.
Return

- `cfix` (`double`) – Fixed term in the objective.

`cj` = `Task.getcj(j)`
Obtains one coefficient of c.

www.EngineeringBooksPdf.com

Parameters

- [in] j (int) – Index of the variable for which c coefficient should be obtained.

Return

- cj (double) – The value of c_j.

bk, bl, bu = Task.getconbound(i)
Obtains bound information for one constraint.

Parameters

- [in] i (int) – Index of the constraint for which the bound information should be obtained.

Return

- bk (int) – Bound keys.
- bl (double) – Values for lower bounds.
- bu (double) – Values for upper bounds.

Task.getconboundslice(first, last, bk, bl, bu)
Obtains bounds information for a slice of the constraints.

Parameters

- [in] first (int) – First index in the sequence.
- [in] last (int) – Last index plus 1 in the sequence.
- [out] bk (*boundkey*) – Bound keys.
- [out] bl (double[]) – Values for lower bounds.
- [out] bu (double[]) – Values for upper bounds.

ct, conepar, nummem = Task.getcone(k, submem)
Obtains a conic constraint.

Parameters

- [in] k (int) – Index of the cone constraint.
- [out] submem (int[]) – Variable subscripts of the members in the cone.

Return

- ct (int) – Specifies the type of the cone.
- conepar (double) – This argument is currently not used. It can be set to 0
- nummem (int) – Number of member variables in the cone.

ct, conepar, nummem = Task.getconeinfo(k)
Obtains information about a conic constraint.

Parameters

- [in] k (int) – Index of the conic constraint.

Return

- ct (int) – Specifies the type of the cone.
- conepar (double) – This argument is currently not used. It can be set to 0
- nummem (int) – Number of member variables in the cone.

name = Task.getconename(i)
Obtains a name of a cone.

Parameters

- [in] i (int) – Index.

Return

- name (str) – Is assigned the required name.

www.EngineeringBooksPdf.com

asgn, index = Task.getconenameindex(somename)
Checks whether the name somename has been assigned to any cone. If it has been assigned to cone, then index of the cone is reported.
Parameters
- [in] somename (str) – The name which should be checked.

Return
- asgn (int) – Is non-zero if the name somename is assigned to a cone.
- index (int) – If the name somename is assigned to a cone, then index is the name of the cone.

len = Task.getconenamelen(i)
Obtains the length of a name of a cone.
Parameters
- [in] i (int) – Index.

Return
- len (int) – Returns the length of the indicated name.

name = Task.getconname(i)
Obtains a name of a constraint.
Parameters
- [in] i (int) – Index.

Return
- name (str) – Is assigned the required name.

asgn, index = Task.getconnameindex(somename)
Checks whether the name somename has been assigned to any constraint. If it has been assigned to constraint, then index of the constraint is reported.
Parameters
- [in] somename (str) – The name which should be checked.

Return
- asgn (int) – Is non-zero if the name somename is assigned to a constraint.
- index (int) – If the name somename is assigned to a constraint, then index is the name of the constraint.

len = Task.getconnamelen(i)
Obtains the length of a name of a constraint variable.
Parameters
- [in] i (int) – Index.

Return
- len (int) – Returns the length of the indicated name.

Task.getcslice(first, last, c)
Obtains a sequence of elements in c.
Parameters
- [in] first (int) – First index in the sequence.
- [in] last (int) – Last index plus 1 in the sequence.
- [out] c (double[]) – Linear terms of the objective as a dense vector. The lengths is the number of variables.

dimbarvarj = Task.getdimbarvarj(j)
Obtains the dimension of a symmetric matrix variable.
Parameters
- [in] j (int) – Index of the semidefinite variable whose dimension is requested.

www.EngineeringBooksPdf.com

Return

- dimbarvarj (int) – The dimension of the j'th semidefinite variable.

dvalue = Task.getdouinf(whichdinf)

Obtains a double information item from the task information database.

Parameters

- [in] whichdinf (*dinfitem*) – A double float information item.

Return

- dvalue (double) – The value of the required double information item.

parvalue = Task.getdouparam(param)

Obtains the value of a double parameter.

Parameters

- [in] param (*dparam*) – Which parameter.

Return

- parvalue (double) – Parameter value.

dualobj = Task.getdualobj(whichsol)

Computes the dual objective value associated with the solution. Note if the solution is a primal inf

Moreover, since there is no dual solution associated with integer solution, then an error will be reported if the dual objective value is requested for the integer solution.

Parameters

- [in] whichsol (*soltype*) – Selects a solution.

Return

- dualobj (double) – Objective value corresponding to the dual solution.

nrmy, nrmslc, nrmsuc, nrmslx, nrmsux, nrmsnx, nrmbars = Task.getdualsolutionnorms(whichsol)

Compute norms of the primal solution.

Parameters

- [in] whichsol (*soltype*) – Selects a solution.

Return

- nrmy (double) – The norm of the y vector.
- nrmslc (double) – The norm of the slc vector.
- nrmsuc (double) – The norm of the suc vector.
- nrmslx (double) – The norm of the slx vector.
- nrmsux (double) – The norm of the sux vector.
- nrmsnx (double) – The norm of the snx vector.
- nrmbars (double) – The norm of the bars vector.

Task.getdviolbarvar(whichsol, sub, viol)

Let $(\bar{S}_j)^*$ be the value of variable $\bar{S}_j$ for the specified solution. Then the dual violation of the solution associated with variable $\bar{S}_j$ is given by

$$\max(-\lambda_{\min}(\bar{S}_j), 0.0).$$

Both when the solution is a certificate of primal infeasibility or when it is dual feasibleness solution the violation should be small.

Parameters

- [in] whichsol (*soltype*) – Selects a solution.
- [in] sub (int[]) – An array of indexes of $\overline{X}$ variables.
- [out] viol (double[]) – viol[k] is violation of the solution for the constraint $\bar{S}_{\mathbf{sub}[k]} \in \mathcal{S}_+$.

www.EngineeringBooksPdf.com

`Task.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 feasibleness solution the violation should be small.

`Parameters`

- `[in]` `whichsol` (*soltype*) – Selects a solution.
- `[in]` `sub` (`int[]`) – An array of indexes of constraints.
- `[out]` `viol` (`double[]`) – `viol[k]` is the violation of dual solution associated with the constraint `sub[k]`.

`Task.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 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 feasibleness solution the violation should be small.

`Parameters`

- `[in]` `whichsol` (*soltype*) – Selects a solution.
- `[in]` `sub` (`int[]`) – An array of indexes of $\overline{X}$ variables.
- `[out]` `viol` (`double[]`) – `viol[k]` violation of the solution associated with `sub[k]`'th dual conic constraint.

`Task.getdviolvar(whichsol, sub, viol)`

The violation of the dual solution associated with the j'th variable is computed as follows

$$\max\left(\rho((s_l^x)_i^*, (b_l^x)_i), \rho((s_u^x)_i^*, -(b_u^x)_i), |\sum_{j=0}^{numcon-1} a_{ij} y_i + (s_l^x)_i^* - (s_u^x)_i^* - \tau c_j|\right)$$

where

$$\rho(x, l) = \begin{cases} -x, & l > -\infty, \\ |x|, & \text{otherwise} \end{cases}$$

$\tau = 0$ if the solution is certificate of dual infeasibility and $\tau = 1$ otherwise. The formula for computing the violation is only shown for linear case but is generalized appropriately for the more general problems.

`Parameters`

- `[in]` `whichsol` (*soltype*) – Selects a solution.
- `[in]` `sub` (`int[]`) – An array of indexes of x variables.
- `[out]` `viol` (`double[]`) – `viol[k]` is the maximal violation of the solution for the constraints $(s_l^x)_{\texttt{sub}[k]} \geq 0$ and $(s_u^x)_{\texttt{sub}[k]} \geq 0$.

`inftask` = `Task.getinfeasiblesubproblem(whichsol)`

Given the solution is a certificate of primal or dual infeasibility then a primal or dual infeasible subproblem is obtained respectively. The subproblem tend to be much smaller than the original

www.EngineeringBooksPdf.com

problem and hence it easier to locate the infeasibility inspecting the subproblem than the original problem.

For the procedure to be useful then it is important to assigning 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 Section *14*.

Parameters

- [in] `whichsol` (*soltype*) – Which solution to use when determining the infeasible subproblem.

Return

- `inftask` (`Task`) – A new task containing the infeasible subproblem.

`ivalue = Task.getintinf(whichiinf)`

Obtains an integer information item from the task information database.

Parameters

- [in] `whichiinf` (*iinfitem*) – Specifies an information item.

Return

- `ivalue` (`int`) – The value of the required integer information item.

`parvalue = Task.getintparam(param)`

Obtains the value of an integer parameter.

Parameters

- [in] `param` (*iparam*) – Which parameter.

Return

- `parvalue` (`int`) – Parameter value.

`lenbarvarj = Task.getlenbarvarj(j)`

Obtains the length of the jth semidefinite variable i.e. the number of elements in the triangular part.

Parameters

- [in] `j` (`int`) – Index of the semidefinite variable whose length if requested.

Return

- `lenbarvarj` (`long`) – Number of scalar elements in the lower triangular part of the semidefinite variable.

`ivalue = Task.getlintinf(whichliinf)`

Obtains an integer information item from the task information database.

Parameters

- [in] `whichliinf` (*liinfitem*) – Specifies an information item.

Return

- `ivalue` (`long`) – The value of the required integer information item.

`maxnumanz = Task.getmaxnumanz()`

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` (`long`) – Number of preallocated non-zero linear matrix elements.

`maxnumbarvar = Task.getmaxnumbarvar()`

Obtains maximum number of symmetric matrix variables that is reservered room for.

Return

- `maxnumbarvar` (`int`) – Maximum number of symmetric matrix variables currently that is reservered room for.

www.EngineeringBooksPdf.com

maxnumcon = Task.getmaxnumcon()
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.

maxnumcone = Task.getmaxnumcone()
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.

maxnumqnz = Task.getmaxnumqnz()
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 (long) – Number of non-zero elements preallocated in quadratic coefficient matrices.

maxnumvar = Task.getmaxnumvar()
Obtains the number of preallocated variables in the optimization task. When this number of variables is reached **MOSEK** will automatically allocate more space for constraints.
Return

- maxnumvar (int) – Number of preallocated variables in the optimization task.

meminuse, maxmemuse = Task.getmemusage()
Obtains information about the amount of memory used by a task.
Return

- meminuse (long) – Amount of memory currently used by the task.
- maxmemuse (long) – Maximum amount of memory used by the task until now.

numanz = Task.getnumanz()
Obtains the number of non-zeros in A.
Return

- numanz (int) – Number of non-zero elements in the linear constraint matrix.

numanz = Task.getnumanz64()
Obtains the number of non-zeros in A.
Return

- numanz (long) – Number of non-zero elements in the linear constraint matrix.

num = Task.getnumbarablocktriplets()
Obtains an upper bound on the number of elements in the block triplet form of $\overline{A}$.
Return

- num (long) – Number elements in the block triplet form of $\overline{A}$.

nz = Task.getnumbaranz()
Get the number of nonzero elements in $\overline{A}$.
Return

- nz (long) – The number of nonzero elements in $\overline{A}$ i.e. the number of $\bar{a}_{ij}$ elements that is nonzero.

num = Task.getnumbarcblocktriplets()
Obtains an upper bound on the number of elements in the block triplet form of $\overline{C}$.
Return

- num (long) – An upper bound on the number elements in the block trip let form of $\bar{c}$.

www.EngineeringBooksPdf.com

`nz` = `Task.getnumbarcnz()`
Obtains the number of nonzero elements in $\bar{c}$.
`Return`

- `nz` (`long`) – The number of nonzeros in $\bar{c}$ i.e. the number of elements $\bar{c}_j$ that is different from 0.

`numbarvar` = `Task.getnumbarvar()`
Obtains the number of semidefinite variables.
`Return`

- `numbarvar` (`int`) – Number of semidefinite variable in the problem.

`numcon` = `Task.getnumcon()`
Obtains the number of constraints.
`Return`

- `numcon` (`int`) – Number of constraints.

`numcone` = `Task.getnumcone()`
Obtains the number of cones.
`Return`

- `numcone` (`int`) – Number conic constraints.

`nummem` = `Task.getnumconemem(k)`
Obtains the number of members in a cone.
`Parameters`

- `[in]` `k` (`int`) – Index of the cone.

`Return`

- `nummem` (`int`) – Number of member variables in the cone.

`numintvar` = `Task.getnumintvar()`
Obtains the number of integer-constrained variables.
`Return`

- `numintvar` (`int`) – Number of integer variables.

`numparam` = `Task.getnumparam(partype)`
Obtains the number of parameters of a given type.
`Parameters`

- `[in]` `partype` (*parametertype*) – Parameter type.

`Return`

- `numparam` (`int`) – Identical to the number of parameters of the type `partype`.

`numqcnz` = `Task.getnumqconknz(k)`
Obtains the number of non-zero quadratic terms in a constraint.
`Parameters`

- `[in]` `k` (`int`) – Index of the constraint for which the number quadratic terms should be obtained.

`Return`

- `numqcnz` (`long`) – Number of quadratic terms.

`numqonz` = `Task.getnumqobjnz()`
Obtains the number of non-zero quadratic terms in the objective.
`Return`

- `numqonz` (`long`) – Number of non-zero elements in the quadratic objective terms.

`num` = `Task.getnumsymmat()`
Get the number of symmetric matrixes stored in the vector E.
`Return`

www.EngineeringBooksPdf.com

- **num** (long) – Returns the number of symmetric sparse matrixes.

numvar = Task.getnumvar()
Obtains the number of variables.
Return

- **numvar** (int) – Number of variables.

objname = Task.getobjname()
Obtains the name assigned to the objective function.
Return

- **objname** (str) – Assigned the objective name.

len = Task.getobjnamelen()
Obtains the length of the name assigned to the objective function.
Return

- **len** (int) – Assigned the length of the objective name.

sense = Task.getobjsense()
Gets the objective sense of the task.
Return

- **sense** (int) – The returned objective sense.

primalobj = Task.getprimalobj(whichsol)
Computes the primal objective value for the desired solution. Note if the solution is an infeasibility certificate, then the fixed term in the objective is not included.
Parameters

- **[in] whichsol** (*soltype*) – Selects a solution.

Return

- **primalobj** (double) – Objective value corresponding to the primal solution.

nrmxc, nrmxx, nrmbarx = Task.getprimalsolutionnorms(whichsol)
Compute norms of the primal solution.
Parameters

- **[in] whichsol** (*soltype*) – Selects a solution.

Return

- **nrmxc** (double) – The norm of xc vector.
- **nrmxx** (double) – The norm of xx vector.
- **nrmbarx** (double) – The norm of barx vector.

probtype = Task.getprobtype()
Obtains the problem type.
Return

- **probtype** (int) – The problem type.

prosta = Task.getprosta(whichsol)
Obtains the problem status.
Parameters

- **[in] whichsol** (*soltype*) – Selects a solution.

Return

- **prosta** (int) – Problem status.

Task.getpviolbarvar(whichsol, sub, viol)
Let $(\overline{X}_j)^*$ be the value of variable $\overline{X}_j$ for the specified solution. Then the primal violation of the solution associated with variable $\overline{X}_j$ is given by

$$\max(-\lambda_{\min}(\overline{X}_j), 0.0).$$

www.EngineeringBooksPdf.com

Parameters

- [in] whichsol (*soltype*) – Selects a solution.
- [in] sub (int[]) – An array of indexes of $\overline{X}$ variables.
- [out] viol (double[]) – viol[k] is how much the solution violate the constraint $\overline{X}_{\texttt{sub}[k]} \in \mathcal{S}_+$.

Task.getpviolcon(whichsol, sub, viol)

The primal violation of the solution associated of constraint is computed by

$$\max(l_i^c\tau - (x_i^c)^*), (x_i^c)^*\tau - u_i^c\tau, |\sum_{j=0}^{numvar-1} a_{ij}x_j^* - x_i^c|)$$

where τ is defined as follows. If the solution is a certificate of dual infeasibility, then $\tau = 0$ and otherwise $\tau = 1$. Both when the solution is a valid certificate of dual infeasibility or when it is primal feasibleness solution the violation should be small. The above is only shown for linear case but is appropriately generalized for the other cases.

Parameters

- [in] whichsol (*soltype*) – Selects a solution.
- [in] sub (int[]) – An array of indexes of constraints.
- [out] viol (double[]) – viol[k] associated with the solution for the sub[k]'th constraint.

Task.getpviolcones(whichsol, sub, viol)

Let x^* be the value of variable x for the specified solution. For simplicity let us assume that x is a member of 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 a primal feasibleness solution the violation should be small.

Parameters

- [in] whichsol (*soltype*) – Selects a solution.
- [in] sub (int[]) – An array of indexes of $\overline{X}$ variables.
- [out] viol (double[]) – viol[k] violation of the solution associated with sub[k]'th conic constraint.

Task.getpviolvar(whichsol, sub, viol)

Let x_j^* be the value of variable x_j for the specified solution. Then the primal violation of the solution associated with variable x_j is given by

$$\max(l_j^x\tau - x_j^*, x_j^* - u_j^x\tau).$$

where τ is defined as follows. If the solution is a certificate of dual infeasibility, then $\tau = 0$ and otherwise $\tau = 1$. Both when the solution is a valid certificate of dual infeasibility or when it is primal feasibleness solution the violation should be small.

Parameters

- [in] whichsol (*soltype*) – Selects a solution.
- [in] sub (int[]) – An array of indexes of x variables.
- [out] viol (double[]) – viol[k] is the violation associated the solution for variable x_j.

numqcnz = Task.getqconk(k, qcsubi, qcsubj, qcval)

Obtains all the quadratic terms in a constraint. The quadratic terms are stored sequentially qcsubi, qcsubj, and qcval.

Parameters

www.EngineeringBooksPdf.com

- [in] k (int) – Which constraint.
- [out] qcsubi (int[]) – Row subscripts for quadratic constraint matrix.
- [out] qcsubj (int[]) – Column subscripts for quadratic constraint matrix.
- [out] qcval (double[]) – Quadratic constraint coefficient values.

Return

- numqcnz (long) – Number of quadratic terms.

numqonz = Task.getqobj(qosubi, qosubj, qoval)

Obtains the quadratic terms in the objective. The required quadratic terms are stored sequentially in qosubi, qosubj, and qoval.

Parameters

- [out] qosubi (int[]) – Row subscripts for quadratic objective coefficients.
- [out] qosubj (int[]) – Column subscripts for quadratic objective coefficients.
- [out] qoval (double[]) – Quadratic objective coefficient values.

Return

- numqonz (long) – Number of non-zero elements in the quadratic objective terms.

qoij = Task.getqobjij(i, j)

Obtains one coefficient q^o_{ij} in the quadratic term of the objective.

Parameters

- [in] i (int) – Row index of the coefficient.
- [in] j (int) – Column index of coefficient.

Return

- qoij (double) – The required coefficient.

Task.getreducedcosts(whichsol, first, last, redcosts)

Computes the reduced costs for a sequence of variables and return them in the variable redcosts i.e.

$$\texttt{redcosts}[j-\texttt{first}] = (s^x_l)_j - (s^x_u)_j,\ j = \texttt{first}, \ldots, last-1 \tag{16.2}$$

Parameters

- [in] whichsol (*soltype*) – Selects a solution.
- [in] first (int) – See formula (16.2) for the definition.
- [in] last (int) – See formula (16.2) for the definition.
- [out] redcosts (double[]) – The reduced costs in the required sequence of variables are stored sequentially in redcosts starting at redcosts[\idxbeg].

Task.getskc(whichsol, skc)

Obtains the status keys for the constraints.

Parameters

- [in] whichsol (*soltype*) – Selects a solution.
- [out] skc (*stakey*) – Status keys for the constraints.

Task.getskcslice(whichsol, first, last, skc)

Obtains the status keys for the constraints.

Parameters

- [in] whichsol (*soltype*) – Selects a solution.
- [in] first (int) – First index in the sequence.
- [in] last (int) – Last index plus 1 in the sequence.

www.EngineeringBooksPdf.com

- [out] skc (*stakey*) – Status keys for the constraints.

Task.getskx(whichsol, skx)
Obtains the status keys for the scalar variables.
Parameters

- [in] whichsol (*soltype*) – Selects a solution.
- [out] skx (*stakey*) – Status keys for the variables.

Task.getskxslice(whichsol, first, last, skx)
Obtains the status keys for the variables.
Parameters

- [in] whichsol (*soltype*) – Selects a solution.
- [in] first (int) – First index in the sequence.
- [in] last (int) – Last index plus 1 in the sequence.
- [out] skx (*stakey*) – Status keys for the variables.

Task.getslc(whichsol, slc)
Obtains the s_l^c vector for a solution.
Parameters

- [in] whichsol (*soltype*) – Selects a solution.
- [out] slc (double[]) – The s_l^c vector.

Task.getslcslice(whichsol, first, last, slc)
Obtains a slice of the s_l^c vector for a solution.
Parameters

- [in] whichsol (*soltype*) – Selects a solution.
- [in] first (int) – First index in the sequence.
- [in] last (int) – Last index plus 1 in the sequence.
- [out] slc (double[]) – Dual variables corresponding to the lower bounds on the constraints.

Task.getslx(whichsol, slx)
Obtains the s_l^x vector for a solution.
Parameters

- [in] whichsol (*soltype*) – Selects a solution.
- [out] slx (double[]) – The s_l^x vector.

Task.getslxslice(whichsol, first, last, slx)
Obtains a slice of the s_l^x vector for a solution.
Parameters

- [in] whichsol (*soltype*) – Selects a solution.
- [in] first (int) – First index in the sequence.
- [in] last (int) – Last index plus 1 in the sequence.
- [out] slx (double[]) – Dual variables corresponding to the lower bounds on the variables.

Task.getsnx(whichsol, snx)
Obtains the s_n^x vector for a solution.
Parameters

- [in] whichsol (*soltype*) – Selects a solution.
- [out] snx (double[]) – The s_n^x vector.

Task.getsnxslice(whichsol, first, last, snx)
Obtains a slice of the s_n^x vector for a solution.

www.EngineeringBooksPdf.com

Parameters

- [in] whichsol (*soltype*) – Selects a solution.
- [in] first (int) – First index in the sequence.
- [in] last (int) – Last index plus 1 in the sequence.
- [out] snx (double[]) – Dual variables corresponding to the conic constraints on the variables.

solsta = Task.getsolsta(whichsol)

Obtains the solution status.

Parameters

- [in] whichsol (*soltype*) – Selects a solution.

Return

- solsta (int) – Solution status.

prosta, solsta = Task.getsolution(whichsol, skc, skx, skn, xc, xx, y, slc, suc, slx, sux, snx)

Obtains the complete solution.

Consider the case of linear programming. The primal problem is given by

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

and the corresponding dual problem is

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

In this case the mapping between variables and arguments to the function is as follows:

- xx : Corresponds to variable x.
- y : Corresponds to variable y.
- slc: Corresponds to variable s_l^c.
- suc: Corresponds to variable s_u^c.
- slx: Corresponds to variable s_l^x.
- sux: Corresponds to variable s_u^x.
- xc : Corresponds to Ax.

The meaning of the values returned by this function depend on the *solution status* returned in the argument solsta. The most important possible values of solsta are:

- *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.

Parameters

- [in] whichsol (*soltype*) – Selects a solution.

www.EngineeringBooksPdf.com

- [out] skc (*stakey*) – Status keys for the constraints.
- [out] skx (*stakey*) – Status keys for the variables.
- [out] skn (*stakey*) – Status keys for the conic constraints.
- [out] xc (double[]) – Primal constraint solution.
- [out] xx (double[]) – Primal variable solution.
- [out] y (double[]) – Vector of dual variables corresponding to the constraints.
- [out] slc (double[]) – Dual variables corresponding to the lower bounds on the constraints.
- [out] suc (double[]) – Dual variables corresponding to the upper bounds on the constraints.
- [out] slx (double[]) – Dual variables corresponding to the lower bounds on the variables.
- [out] sux (double[]) – Dual variables corresponding to the upper bounds on the variables.
- [out] snx (double[]) – Dual variables corresponding to the conic constraints on the variables.

Return

- prosta (int) – Problem status.
- solsta (int) – Solution status.

sk, x, sl, su, sn = Task.getsolutioni(accmode, i, whichsol)
Obtains the primal and dual solution information for a single constraint or variable.

Parameters

- [in] accmode (*accmode*) – If set to *accmode.con* the solution information for a constraint is retrieved. Otherwise for a variable.
- [in] i (int) – Index of the constraint or variable.
- [in] whichsol (*soltype*) – Selects a solution.

Return

- sk (int) – Status key of the constraint of variable.
- x (double) – Solution value of the primal variable.
- sl (double) – Solution value of the dual variable associated with the lower bound.
- su (double) – Solution value of the dual variable associated with the upper bound.
- sn (double) – Solution value of the dual variable associated with the cone constraint.

pobj, pviolcon, pviolvar, pviolbarvar, pviolcone, pviolitg, dobj, dviolcon, dviolvar, dviolbarvar,
Obtains information about a solution.

Parameters

- [in] whichsol (*soltype*) – Selects a solution.

Return

- pobj (double) – The primal objective value as computed by *task.getprimalobj*.
- pviolcon (double) – Maximal primal violation of the solution associated with the x^c variables where the violations are computed by *task.getpviolcon*.
- pviolvar (double) – Maximal primal violation of the solution for the x^x variables where the violations are computed by *task.getpviolvar*.
- pviolbarvar (double) – Maximal primal violation of solution for the $\overline{X}$ variables where the violations are computed by *task.getpviolbarvar*.
- pviolcone (double) – Maximal primal violation of solution for the conic constraints where the violations are computed by *task.getpviolcones*.

www.EngineeringBooksPdf.com

- **pviolitg** (**double**) – Maximal violation in the integer constraints. The violation for an integer constrained 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 the basic solutions.

- **dobj** (**double**) – Dual objective value as computed as computed by *task.getdualobj*.

- **dviolcon** (**double**) – Maximal violation of the dual solution associated with the x^c variable as computed by as computed by *task.getdviolcon*.

- **dviolvar** (**double**) – Maximal violation of the dual solution associated with the x variable as computed by as computed by *task.getdviolvar*.

- **dviolbarvar** (**double**) – Maximal violation of the dual solution associated with the $\bar{s}$ variable as computed by as computed by *task.getdviolbarvar*.

- **dviolcone** (**double**) – Maximal violation of the dual solution associated with the dual conic constraints as computed by *task.getdviolcones*.

Task.getsolutionslice(whichsol, solitem, first, last, values)

Obtains a slice of the solution.

Consider the case of linear programming. The primal problem is given by

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

and the corresponding dual problem is

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

The **solitem** argument determines which part of the solution is returned:

- *solitem.xx* : The variable **values** return x.
- *solitem.y* : The variable **values** return y.
- *solitem.slc* : The variable **values** return s_l^c.
- *solitem.suc* : The variable **values** return s_u^c.
- *solitem.slx* : The variable **values** return s_l^x.
- *solitem.sux* : The variable **values** return s_u^x.

A conic optimization problem has the same primal variables as in the linear case. Recall that the dual of a conic optimization problem is given by:

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

This introduces one additional dual variable s_n^x. This variable can be acceded by selecting **solitem** as *solitem.snx*.

The meaning of the values returned by this function also depends on the *solution status* which can be obtained with *task.getsolsta*. Depending on the solution status **value** will be:

www.EngineeringBooksPdf.com

- *solsta.optimal* A part of the optimal solution satisfying the optimality criteria for continuous problems.
- *solsta.integer_optimal* A part of the optimal solution satisfying the optimality criteria for integer problems.
- *solsta.prim_feas* A part of the solution satisfying the feasibility criteria.
- *solsta.prim_infeas_cer* A part of the primal certificate of infeasibility.
- *solsta.dual_infeas_cer* A part of the dual certificate of infeasibility.

Parameters

- [in] whichsol (*soltype*) – Selects a solution.
- [in] solitem (*solitem*) – Which part of the solution is required.
- [in] first (int) – Index of the first value in the slice.
- [in] last (int) – value of the last index+1 in the slice, e.g. if $xx[5,\ldots,9]$ is required last should be 10.
- [out] values (double[]) – The values in the required sequence are stored sequentially in values starting at values[\idxbeg].

Task.getsparsesymmat(idx, subi, subj, valij)
Get a single symmetric matrix from the matrix store.

Parameters

- [in] idx (long) – Index of the matrix to get.
- [out] subi (int[]) – Row subscripts of the matrix non-zero elements.
- [out] subj (int[]) – Column subscripts of the matrix non-zero elements.
- [out] valij (double[]) – Coefficients of the matrix non-zero elements.

len, parvalue = Task.getstrparam(param)
Obtains the value of a string parameter.

Parameters

- [in] param (*sparam*) – Which parameter.

Return

- len (int) – The length of the parameter value.
- parvalue (str) – If this is not NULL, the parameter value is stored here.

len = Task.getstrparamlen(param)
Obtains the length of a string parameter.

Parameters

- [in] param (*sparam*) – Which parameter.

Return

- len (int) – The length of the parameter value.

Task.getsuc(whichsol, suc)
Obtains the s_u^c vector for a solution.

Parameters

- [in] whichsol (*soltype*) – Selects a solution.
- [out] suc (double[]) – The s_u^c vector.

Task.getsucslice(whichsol, first, last, suc)
Obtains a slice of the s_u^c vector for a solution.

Parameters

- [in] whichsol (*soltype*) – Selects a solution.
- [in] first (int) – First index in the sequence.

www.EngineeringBooksPdf.com

- [in] last (int) – Last index plus 1 in the sequence.
- [out] suc (double[]) – Dual variables corresponding to the upper bounds on the constraints.

Task.getsux(whichsol, sux)
Obtains the s_u^x vector for a solution.
Parameters

- [in] whichsol (*soltype*) – Selects a solution.
- [out] sux (double[]) – The s_u^x vector.

Task.getsuxslice(whichsol, first, last, sux)
Obtains a slice of the s_u^x vector for a solution.
Parameters

- [in] whichsol (*soltype*) – Selects a solution.
- [in] first (int) – First index in the sequence.
- [in] last (int) – Last index plus 1 in the sequence.
- [out] sux (double[]) – Dual variables corresponding to the upper bounds on the variables.

dim, nz, type = Task.getsymmatinfo(idx)
MOSEK maintains a vector denoted by E of symmetric data matrixes. This function makes it possible to obtain important information about an data matrix in E.
Parameters

- [in] idx (long) – Index of the matrix that is requested information about.

Return

- dim (int) – Returns the dimension of the requested matrix.
- nz (long) – Returns the number of non-zeros in the requested matrix.
- type (int) – Returns the type of the requested matrix.

taskname = Task.gettaskname()
Obtains the name assigned to the task.
Return

- taskname (str) – Is assigned the task name.

len = Task.gettasknamelen()
Obtains the length the task name.
Return

- len (int) – Returns the length of the task name.

bk, bl, bu = Task.getvarbound(i)
Obtains bound information for one variable.
Parameters

- [in] i (int) – Index of the variable for which the bound information should be obtained.

Return

- bk (int) – Bound keys.
- bl (double) – Values for lower bounds.
- bu (double) – Values for upper bounds.

Task.getvarboundslice(first, last, bk, bl, bu)
Obtains bounds information for a slice of the variables.
Parameters

- [in] first (int) – First index in the sequence.
- [in] last (int) – Last index plus 1 in the sequence.
- [out] bk (*boundkey*) – Bound keys.

www.EngineeringBooksPdf.com

- [out] bl (double[]) – Values for lower bounds.
- [out] bu (double[]) – Values for upper bounds.

name = Task.getvarname(j)
Obtains a name of a variable.
Parameters

- [in] j (int) – Index.

Return

- name (str) – Returns the required name.

asgn, index = Task.getvarnameindex(somename)
Checks whether the name somename has been assigned to any variable. If it has been assigned to variable, then index of the variable is reported.
Parameters

- [in] somename (str) – The name which should be checked.

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 name of the variable.

len = Task.getvarnamelen(i)
Obtains the length of a name of a variable variable.
Parameters

- [in] i (int) – Index.

Return

- len (int) – Returns the length of the indicated name.

vartype = Task.getvartype(j)
Gets the variable type of one variable.
Parameters

- [in] j (int) – Index of the variable.

Return

- vartype (int) – Variable type of variable j.

Task.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

- [in] subj (int[]) – A list of variable indexes.
- [out] vartype (*variabletype*) – The variables types corresponding to the variables specified by subj.

Task.getxc(whichsol, xc)
Obtains the x^c vector for a solution.
Parameters

- [in] whichsol (*soltype*) – Selects a solution.
- [out] xc (double[]) – The x^c vector.

Task.getxcslice(whichsol, first, last, xc)
Obtains a slice of the x^c vector for a solution.
Parameters

- [in] whichsol (*soltype*) – Selects a solution.
- [in] first (int) – First index in the sequence.

www.EngineeringBooksPdf.com

- [in] `last` (`int`) – Last index plus 1 in the sequence.
- [out] `xc` (`double[]`) – Primal constraint solution.

`Task.getxx(whichsol, xx)`
Obtains the x^x vector for a solution.
Parameters

- [in] `whichsol` (*soltype*) – Selects a solution.
- [out] `xx` (`double[]`) – The x^x vector.

`Task.getxxslice(whichsol, first, last, xx)`
Obtains a slice of the x^x vector for a solution.
Parameters

- [in] `whichsol` (*soltype*) – Selects a solution.
- [in] `first` (`int`) – First index in the sequence.
- [in] `last` (`int`) – Last index plus 1 in the sequence.
- [out] `xx` (`double[]`) – Primal variable solution.

`Task.gety(whichsol, y)`
Obtains the y vector for a solution.
Parameters

- [in] `whichsol` (*soltype*) – Selects a solution.
- [out] `y` (`double[]`) – The y vector.

`Task.getyslice(whichsol, first, last, y)`
Obtains a slice of the y vector for a solution.
Parameters

- [in] `whichsol` (*soltype*) – Selects a solution.
- [in] `first` (`int`) – First index in the sequence.
- [in] `last` (`int`) – Last index plus 1 in the sequence.
- [out] `y` (`double[]`) – Vector of dual variables corresponding to the constraints.

`Task.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 (1615)* is reported.
Parameters

- [out] `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.

`Task.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 *16.1.3.2*.

For an explained code example see Section *3.2* and Section *16.1.3*.
Parameters

www.EngineeringBooksPdf.com

- [in] maxnumcon (int) – Number of preallocated constraints in the optimization task.
- [in] maxnumvar (int) – Number of preallocated variables in the optimization task.
- [in] c (double[]) – Linear terms of the objective as a dense vector. The lengths is the number of variables.
- [in] cfix (double) – Fixed term in the objective.
- [in] aptrb (long[]) – Row or column end pointers.
- [in] aptre (long[]) – Row or column start pointers.
- [in] asub (int[]) – Coefficient subscripts.
- [in] aval (double[]) – Coefficient values.
- [in] bkc (*boundkey*) – Bound keys for the constraints.
- [in] blc (double[]) – Lower bounds for the constraints.
- [in] buc (double[]) – Upper bounds for the constraints.
- [in] bkx (*boundkey*) – Bound keys for the variables.
- [in] blx (double[]) – Lower bounds for the variables.
- [in] bux (double[]) – Upper bounds for the variables.

param = Task.isdouparname(parname)
Checks whether parname is a valid double parameter name.
Parameters

- [in] parname (str) – Parameter name.

Return

- param (int) – Which parameter.

param = Task.isintparname(parname)
Checks whether parname is a valid integer parameter name.
Parameters

- [in] parname (str) – Parameter name.

Return

- param (int) – Which parameter.

param = Task.isstrparname(parname)
Checks whether parname is a valid string parameter name.
Parameters

- [in] parname (str) – Parameter name.

Return

- param (int) – Which parameter.

Task.linkfiletostream(whichstream, filename, append)
Directs all output from a task stream to a file.
Parameters

- [in] whichstream (*streamtype*) – Index of the stream.
- [in] filename (str) – The name of the file where text from the stream defined by whichstream is written.
- [in] append (int) – If this argument is 0 the output file will be overwritten, otherwise text is append to the output file.

Task.onesolutionsummary(whichstream, whichsol)
Prints a short summary for a specified solution.
Parameters

www.EngineeringBooksPdf.com

- [in] `whichstream` (*streamtype*) – Index of the stream.
- [in] `whichsol` (*soltype*) – Selects a solution.

`trmcode = Task.optimize()`

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*.

This function returns errors on the left hand side. Warnings are not returned and termination codes are returned in the separate argument `trmcode`.

`Return`

- `trmcode` (`int`) – Is either *rescodetype.ok* or a termination response code.

`trmcode = Task.optimizermt(server, port)`

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`

- [in] `server` (`str`) – Name or IP address of the solver server
- [in] `port` (`str`) – Network port of solver service

`Return`

- `trmcode` (`int`) – Is either *rescodetype.ok* or a termination response code.

`Task.optimizersummary(whichstream)`

Prints a short summary with optimizer statistics for last optimization.

`Parameters`

- [in] `whichstream` (*streamtype*) – Index of the stream.

`Task.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 relaxation to the bounds on the constraints and variables. Observe the function only repairs the problem but does not compute an optimal solution to the repaired problem. If an optimal solution is required the problem should be optimized after the repair.

The function is applicable to linear and conic problems possibly having integer constrained variables.

Observe that when computing the minimal weighted relaxation then the termination tolerance specified by the parameters of the task is employed. For instance the parameter *iparam.mio_mode* can be used 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 bounds on the constraints and variables. If this is not a desired feature, then apply the function to a cloned task.

`Parameters`

- [in] `wlc` (`double[]`) – $(w_l^c)_i$ is the weight associated with relaxing the lower bound on constraint i. If the weight is negative, then the lower bound is not relaxed. Moreover, if the argument is NULL, then all the weights are assumed to be 1.
- [in] `wuc` (`double[]`) – $(w_u^c)_i$ is the weight associated with relaxing the upper bound on constraint i. If the weight is negative, then the upper bound is not relaxed. Moreover, if the argument is NULL, then all the weights are assumed to be 1.
- [in] `wlx` (`double[]`) – $(w_l^x)_j$ is the weight associated with relaxing the upper bound on constraint 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.

www.EngineeringBooksPdf.com

- [in] `wux` (`double[]`) – $(w_l^x)_i$ 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.

`Task.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 and the definitions of *shadow price* and *linearity interval* see Section *15*.

The constraints for which sensitivity analysis is performed are given by the data structures:

1. `subi` Index of constraint to analyze.
2. `marki` Indicate for which bound of constraint `subi[i]` sensitivity analysis is performed. If `marki[i]` = *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.

Consider the problem:

$$\begin{array}{lrcl} \text{minimize} & x_1 + x_2 & & \\ \text{subject to} -1 \leq & x_1 - x_2 & \leq & 1, \\ & x_1 & = & 0, \\ & x_1 \geq 0, x_2 \geq 0 & & \end{array}$$

Suppose that

- `numi = 1;`
- `subi = [0];`
- `marki` = [*mark.up*]

then

`leftpricei[0]`, `rightpricei[0]`, `leftrangei[0]` and `rightrangei[0]` will contain the sensitivity information for the upper bound on constraint 0 given by the expression:

$$x_1 - x_2 \leq 1$$

Similarly, the variables for which to perform sensitivity analysis are given by the structures:

1. `subj` Index of variables to analyze.
2. `markj` Indicate for which bound of variable `subi[j]` sensitivity analysis is performed. If `markj[j]` = *mark.up* the upper bound of constraint `subi[j]` is analyzed, and if `markj[j]` = *mark.lo* the lower bound is analyzed.
3. If `subi[j]` is an equality constraint, either *mark.lo* or *mark.up* can be used to select the constraint for sensitivity analysis.

For an example, please see Section *15.1.4*.

The type of sensitivity analysis to be performed (basis or optimal partition) is controlled by the parameter *iparam.sensitivity_type*.

`Parameters`

- [in] `subi` (`int[]`) – Indexes of bounds on constraints to analyze.
- [in] `marki` (*mark*) – The value of `marki[i]` specifies for which bound (upper or lower) on constraint `subi[i]` sensitivity analysis should be performed.
- [in] `subj` (`int[]`) – Indexes of bounds on variables to analyze.
- [in] `markj` (*mark*) – The value of `markj[j]` specifies for which bound (upper or lower) on variable `subj[j]` sensitivity analysis should be performed.

www.EngineeringBooksPdf.com

- [out] leftpricei (double[]) – leftpricei[i] is the left shadow price for the upper/lower bound (indicated by marki[i]) of the constraint with index subi[i].
- [out] rightpricei (double[]) – rightpricei[i] is the right shadow price for the upper/lower bound (indicated by marki[i]) of the constraint with index subi[i].
- [out] leftrangei (double[]) – leftrangei[i] is the left range for the upper/lower bound (indicated by marki[i]) of the constraint with index subi[i].
- [out] rightrangei (double[]) – rightrangei[i] is the right range for the upper/lower bound (indicated by marki[i]) of the constraint with index subi[i].
- [out] leftpricej (double[]) – leftpricej[j] is the left shadow price for the upper/lower bound (indicated by marki[j]) on variable subj[j].
- [out] rightpricej (double[]) –

  rightpricej[j] **is the right shadow price for the upper/lower bound (indicated by** marki[j]**)** on variable subj[j].
- [out] leftrangej (double[]) –

  leftrangej[j] **is the left range for the upper/lower bound (indicated by** marki[j]**)** on variable subj[j].
- [out] rightrangej (double[]) – rightrangej[j] is the right range for the upper/lower bound (indicated by marki[j]) on variable subj[j].

Task.printdata(whichstream, firsti, lasti, firstj, lastj, firstk, lastk, c, qo, a, qc, bc, bx, vartype, cones)

Prints a part of the problem data to a stream. This function is normally used for debugging purposes only, e.g. to verify that the correct data has been inputted.

Parameters

- [in] whichstream (*streamtype*) – Index of the stream.
- [in] firsti (int) – Index of first constraint for which data should be printed.
- [in] lasti (int) – Index of last constraint plus 1 for which data should be printed.
- [in] firstj (int) – Index of first variable for which data should be printed.
- [in] lastj (int) – Index of last variable plus 1 for which data should be printed.
- [in] firstk (int) – Index of first cone for which data should be printed.
- [in] lastk (int) – Index of last cone plus 1 for which data should be printed.
- [in] c (int) – If non-zero c is printed.
- [in] qo (int) – If non-zero Q^o is printed.
- [in] a (int) – If non-zero A is printed.
- [in] qc (int) – If non-zero Q^k is printed for the relevant constraints.
- [in] bc (int) – If non-zero the constraints bounds are printed.
- [in] bx (int) – If non-zero the variable bounds are printed.
- [in] vartype (int) – If non-zero the variable types are printed.
- [in] cones (int) – If non-zero the conic data is printed.

Task.putacol(j, subj, valj)

Resets all the elements in column j to zero and then do

$$A_{\texttt{subj}[k],\texttt{j}} = \texttt{valj}[k], \quad k = 0, \ldots, \texttt{nzj} - 1.$$

Parameters

- [in] j (int) – Index of column in A.

www.EngineeringBooksPdf.com

- [in] `subj` (`int[]`) – Row indexes of non-zero values in column j of A.
- [in] `valj` (`double[]`) – New non-zero values of column j in A.

`Task.putacollist(sub, ptrb, ptre, asub, aval)`
Replaces all elements in a set of columns of A. The elements are replaced as follows

$$\begin{array}{ll}\texttt{for} & i = 0, \ldots, num - 1 \\ & a_{\texttt{asub}[k],\texttt{sub}[i]} = \texttt{aval}[k], \quad k = \texttt{aptrb}[i], \ldots, \texttt{aptre}[i] - 1.\end{array}$$

Parameters

- [in] `sub` (`int[]`) – Indexes of columns that should be replaced. `comp` should not contain duplicate values.
- [in] `ptrb` (`long[]`) – Array of pointers to the first element in the columns stored in `comp` and `comp`.

  For an explanation of the meaning of `comp` see Section *16.1.3.2*.
- [in] `ptre` (`long[]`) – Array of pointers to the last element plus one in the columns stored in `comp` and `comp`.

  For an explanation of the meaning of `comp` see Section *16.1.3.2*.
- [in] `asub` (`int[]`) – `comp` contains the new variable indexes.
- [in] `aval` (`double[]`) – Coefficient values.

`Task.putacolslice(first, last, ptrb, ptre, asub, aval)`
Replaces all elements in a set of columns of A.
Parameters

- [in] `first` (`int`) – First column in the slice.
- [in] `last` (`int`) – Last column plus one in the slice.
- [in] `ptrb` (`long[]`) – Array of pointers to the first element in the columns stored in `comp` and `comp`.

  For an explanation of the meaning of `comp` see Section *16.1.3.2*.
- [in] `ptre` (`long[]`) – Array of pointers to the last element plus one in the columns stored in `comp` and `comp`.

  For an explanation of the meaning of `comp` see Section *16.1.3.2*.
- [in] `asub` (`int[]`) – `comp` contains the new variable indexes.
- [in] `aval` (`double[]`) – Coefficient values.

`Task.putaij(i, j, aij)`
Changes a coefficient in A using the method

$$a_{\texttt{ij}} = \texttt{aij}.$$

Parameters

- [in] `i` (`int`) – Index of the constraint in which the change should occur.
- [in] `j` (`int`) – Index of the variable in which the change should occur.
- [in] `aij` (`double`) – New coefficient for $a_{i,j}$.

`Task.putaijlist(subi, subj, valij)`
Changes one or more coefficients in A using the method

$$a_{\texttt{subi[k]},\texttt{subj[k]}} = \texttt{valij[k]}, \quad k = 0, \ldots, \texttt{num} - 1.$$

Parameters

www.EngineeringBooksPdf.com

- [in] `subi` (`int[]`) – Constraint indexes in which the change should occur.
- [in] `subj` (`int[]`) – Variable indexes in which the change should occur.
- [in] `valij` (`double[]`) – New coefficient values for $a_{i,j}$.

`Task.putarow(i, subi, vali)`
Resets all the elements in row i to zero and then do

$$A_{\texttt{i},\texttt{subi}[k]} = \texttt{vali}[k], \quad k = 0, \ldots, \texttt{nzi} - 1.$$

Parameters

- [in] `i` (`int`) – Index of row in A.
- [in] `subi` (`int[]`) – Row indexes of non-zero values in row i of A.
- [in] `vali` (`double[]`) – New non-zero values of row i in A.

`Task.putarowlist(sub, ptrb, ptre, asub, aval)`
Replaces all elements in a set of rows of A. The elements are replaced as follows

$$\begin{array}{ll} \texttt{for} & i = \texttt{first}, \ldots, \texttt{last} - 1 \\ & a_{\texttt{sub[i]},\texttt{asub}[k]} = \texttt{aval}[k], \quad k = \texttt{aptrb}[i], \ldots, \texttt{aptre}[i] - 1. \end{array}$$

Parameters

- [in] `sub` (`int[]`) – Indexes of rows or columns that should be replaced. `comp` should not contain duplicate values.
- [in] `ptrb` (`long[]`) – Array of pointers to the first element in the rows stored in `comp` and `comp`.

  For an explanation of the meaning of `comp` see Section *16.1.3.2*.
- [in] `ptre` (`long[]`) – Array of pointers to the last element plus one in the rows stored in `comp` and `comp`.

  For an explanation of the meaning of `comp` see Section *16.1.3.2*.
- [in] `asub` (`int[]`) – `comp` contains the new variable indexes.
- [in] `aval` (`double[]`) – Coefficient values.

`Task.putarowslice(first, last, ptrb, ptre, asub, aval)`
Replaces all elements in a set of rows of A. The elements is replaced as follows

$$\begin{array}{ll} \texttt{for} & i = \texttt{first}, \ldots, \texttt{last} - 1 \\ & a_{\texttt{sub}[i],\texttt{asub}[k]} = \texttt{aval}[k], \quad k = \texttt{aptrb}[i], \ldots, \texttt{aptre}[i] - 1. \end{array}$$

Parameters

- [in] `first` (`int`) – First row in the slice.
- [in] `last` (`int`) – Last row plus one in the slice.
- [in] `ptrb` (`long[]`) – Array of pointers to the first element in the rows stored in `comp` and `comp`.

  For an explanation of the meaning of `comp` see Section *16.1.3.2*.
- [in] `ptre` (`long[]`) – Array of pointers to the last element plus one in the rows stored in `comp` and `comp`.

  For an explanation of the meaning of `comp` see Section *16.1.3.2*.
- [in] `asub` (`int[]`) – `comp` contains the new variable indexes.
- [in] `aval` (`double[]`) – Coefficient values.

`Task.putbarablocktriplet(num, subi, subj, subk, subl, valijkl)`
Inputs the $\overline{A}$ in block triplet form.

www.EngineeringBooksPdf.com

Parameters

- [in] num (long) – Number of elements in the block triplet form.
- [in] subi (int[]) – Constraint index.
- [in] subj (int[]) – Symmetric matrix variable index.
- [in] subk (int[]) – Block row index.
- [in] subl (int[]) – Block column index.
- [in] valijkl (double[]) – The numerical value associated with the block triplet.

Task.putbaraij(i, j, sub, weights)
This function puts one element associated with $\overline{X}_j$ in the $\overline{A}$ matrix.

Each element in the $\overline{A}$ matrix is a weighted sum of symmetric matrixes, i.e. $\overline{A}_{ij}$ is a symmetric matrix with dimensions as $\overline{X}_j$. By default all elements in $\overline{A}$ are 0, so only non-zero elements need be added.

Setting the same elements again will overwrite the earlier entry.

The symmetric matrixes themselves are defined separately using the function *task.appendsparsesymmat*.

Parameters

- [in] i (int) – Row index of $\overline{A}$.
- [in] j (int) – Column index of $\overline{A}$.
- [in] sub (long[]) – See argument weights for an explanation.
- [in] weights (double[]) – weights[k] times sub[k]'th term of E is added to $\overline{A}_{ij}$.

Task.putbarcblocktriplet(num, subj, subk, subl, valjkl)
Inputs the $\overline{C}$ in block triplet form.

Parameters

- [in] num (long) – Number of elements in the block triplet form.
- [in] subj (int[]) – Symmetric matrix variable index.
- [in] subk (int[]) – Block row index.
- [in] subl (int[]) – Block column index.
- [in] valjkl (double[]) – The numerical value associated with the block triplet.

Task.putbarcj(j, sub, weights)
This function puts one element associated with $\overline{X}_j$ in the $\bar{c}$ vector.

Each element in the $\bar{c}$ vector is a weighted sum of symmetric matrixes, i.e. $\bar{c}_j$ is a symmetric matrix with dimensions as $\overline{X}_j$. By default all elements in $\bar{c}$ are 0, so only non-zero elements need be added.

Setting the same elements again will overwrite the earlier entry.

The symmetric matrixes themselves are defined separately using the function *task.appendsparsesymmat*.

Parameters

- [in] j (int) – Index of the element in $\bar{c}$ that should be changed.
- [in] sub (long[]) – sub is list of indexes of those symmetric matrices appearing in sum.
- [in] weights (double[]) – The weights of the terms in the weighted sum that forms c_j.

Task.putbarsj(whichsol, j, barsj)
Sets the dual solution for a semidefinite variable.

Parameters

- [in] whichsol (*soltype*) – Selects a solution.

www.EngineeringBooksPdf.com

- [in] `j` (`int`) – Index of the semidefinite variable.
- [in] `barsj` (`double[]`) – Value of $\bar{s}_j$.

`Task.putbarvarname(j, name)`
Puts the name of a semidefinite variable.
`Parameters`

- [in] `j` (`int`) – Index of the variable.
- [in] `name` (`str`) – The variable name.

`Task.putbarxj(whichsol, j, barxj)`
Sets the primal solution for a semidefinite variable.
`Parameters`

- [in] `whichsol` (*soltype*) – Selects a solution.
- [in] `j` (`int`) – Index of the semidefinite variable.
- [in] `barxj` (`double[]`) – Value of $\overline{X}_j$.

`Task.putbound(accmode, i, bk, bl, bu)`
Changes the bounds for either one constraint or one variable.

If the a 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`

- [in] `accmode` (*accmode*) – Defines whether the bound for a constraint or a variable is changed.
- [in] `i` (`int`) – Index of the constraint or variable.
- [in] `bk` (*boundkey*) – New bound key.
- [in] `bl` (`double`) – New lower bound.
- [in] `bu` (`double`) – New upper bound.

`Task.putboundlist(accmode, sub, bk, bl, bu)`
Changes the bounds for either some constraints or variables. If multiple bound changes are specified for a constraint or a variable, only the last change takes effect.
`Parameters`

- [in] `accmode` (*accmode*) – Defines whether bounds for constraints (*accmode.con*) or variables (*accmode.var*) are changed.
- [in] `sub` (`int[]`) – Subscripts of the bounds that should be changed.
- [in] `bk` (*boundkey*) – Constraint or variable index `sub[t]` is assigned the bound key `bk[t]`.
- [in] `bl` (`double[]`) – Constraint or variable index `sub[t]` is assigned the lower bound `bl[t]`.
- [in] `bu` (`double[]`) – Constraint or variable index `sub[t]` is assigned the upper bound `bu[t]`.

`Task.putboundslice(con, first, last, bk, bl, bu)`
Changes the bounds for a sequence of variables or constraints.
`Parameters`

- [in] `con` (*accmode*) – Defines whether bounds for constraints (*accmode.con*) or variables (*accmode.var*) are changed.
- [in] `first` (`int`) – First index in the sequence.
- [in] `last` (`int`) – Last index plus 1 in the sequence.
- [in] `bk` (*boundkey*) – Bound keys.

www.EngineeringBooksPdf.com

- **[in]** `bl` (`double[]`) – Values for lower bounds.
- **[in]** `bu` (`double[]`) – Values for upper bounds.

`Task.putcfix(cfix)`
Replaces the fixed term in the objective by a new one.
Parameters

- **[in]** `cfix` (`double`) – Fixed term in the objective.

`Task.putcj(j, cj)`
Modifies one coefficient in the linear objective vector c, i.e.

$$c_{\texttt{j}} = \texttt{cj}.$$

Parameters

- **[in]** `j` (`int`) – Index of the variable for which c should be changed.
- **[in]** `cj` (`double`) – New value of c_j.

`Task.putclist(subj, val)`
Modifies elements in the linear term c in the objective using the principle

$$c_{\texttt{subj[t]}} = \texttt{val[t]}, \quad t = 0, \ldots, \texttt{num} - 1.$$

If a variable index is specified multiple times in `subj` only the last entry is used.
Parameters

- **[in]** `subj` (`int[]`) – Index of variables for which c should be changed.
- **[in]** `val` (`double[]`) – New numerical values for coefficients in c that should be modified.

`Task.putconbound(i, bk, bl, bu)`
Changes the bounds for one constraint.

If the a 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

- **[in]** `i` (`int`) – Index of the constraint.
- **[in]** `bk` (*boundkey*) – New bound key.
- **[in]** `bl` (`double`) – New lower bound.
- **[in]** `bu` (`double`) – New upper bound.

`Task.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.
Parameters

- **[in]** `sub` (`int[]`) – List constraints indexes.
- **[in]** `bkc` (*boundkey*) – New bound keys.
- **[in]** `blc` (`double[]`) – New lower bound values.
- **[in]** `buc` (`double[]`) – New upper bound values.

`Task.putconboundslice(first, last, bk, bl, bu)`
Changes the bounds for a slice of the constraints.
Parameters

- **[in]** `first` (`int`) – Index of the first constraint in the slice.
- **[in]** `last` (`int`) – Index of the last constraint in the slice plus 1.
- **[in]** `bk` (*boundkey*) – New bound keys.

www.EngineeringBooksPdf.com

- [in] bl (double[]) – New lower bounds.
- [in] bu (double[]) – New upper bounds.

Task.putcone(k, ct, conepar, submem)
Replaces a conic constraint.
Parameters

- [in] k (int) – Index of the cone.
- [in] ct (*conetype*) – Specifies the type of the cone.
- [in] conepar (double) – This argument is currently not used. It can be set to 0
- [in] submem (int[]) – Variable subscripts of the members in the cone.

Task.putconename(j, name)
Puts the name of a cone.
Parameters

- [in] j (int) – Index of the cone.
- [in] name (str) – The variable name.

Task.putconname(i, name)
Puts the name of a constraint.
Parameters

- [in] i (int) – Index of the constraint.
- [in] name (str) – The variable name.

Task.putcslice(first, last, slice)
Modifies a slice in the linear term c in the objective using the principle

$$c_{\mathtt{j}} = \mathtt{slice}[\mathtt{j} - \mathtt{first}], \quad j = first, .., last - 1$$

Parameters

- [in] first (int) – First element in the slice of c.
- [in] last (int) – Last element plus 1 of the slice in c to be changed.
- [in] slice (double[]) – New numerical values for coefficients in c that should be modified.

Task.putdouparam(param, parvalue)
Sets the value of a double parameter.
Parameters

- [in] param (*dparam*) – Which parameter.
- [in] parvalue (double) – Parameter value.

Task.putintparam(param, parvalue)
Sets the value of an integer parameter.
Parameters

- [in] param (*iparam*) – Which parameter.
- [in] parvalue (int) – Parameter value.

Task.putmaxnumanz(maxnumanz)
MOSEK stores only the non-zero elements in A. Therefore1 **MOSEK** cannot predict how much storage is required to store A. Using this function it is possible to specify the number of non-zeros to preallocate for storing A.

If the number of non-zeros in the problem is known, it is a good idea to set maxnumanz slightly larger than this number, otherwise a rough estimate can be used. In general, if A is inputted in many small chunks, setting this value may speed up the data input phase.

www.EngineeringBooksPdf.com

It is not mandatory to call this function, since **MOSEK** will reallocate internal structures whenever it is necessary.

Observe the function call has no effect if both `maxnumcon` and `maxnumvar` is zero.

`Parameters`

- `[in]` `maxnumanz` (`long`) – New size of the storage reserved for storing A.

`Task.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, since its only function is to give a hint of the amount of data to preallocate for efficiency reasons.

Please note that `maxnumbarvar` must be larger than the current number of variables in the task.

`Parameters`

- `[in]` `maxnumbarvar` (`int`) – The maximum number of semidefinite variables.

`Task.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`

- `[in]` `maxnumcon` (`int`) – Number of preallocated constraints in the optimization task.

`Task.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 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`

- `[in]` `maxnumcone` (`int`) – Number of preallocated conic constraints in the optimization task.

`Task.putmaxnumqnz(maxnumqnz)`

MOSEK stores only the non-zero elements in Q. Therefore, **MOSEK** cannot predict how much storage is required to store Q. Using this function it is possible to specify the number non-zeros to preallocate for storing Q (both objective and constraints).

It may be advantageous to reserve more non-zeros for Q than actually needed since it may improve the internal efficiency of **MOSEK**, however, it is never worthwhile to specify more than the double of the anticipated number of non-zeros in Q.

It is never mandatory to call this function, since its only function is to give a hint of the amount of data to preallocate for efficiency reasons.

`Parameters`

- `[in]` `maxnumqnz` (`long`) – Number of non-zero elements preallocated in quadratic coefficient matrices.

`Task.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 never mandatory to call this function, since its only function is to give a hint of the amount of data to preallocate for efficiency reasons.

Please note that `maxnumvar` must be larger than the current number of variables in the task.

`Parameters`

www.EngineeringBooksPdf.com

- [in] `maxnumvar` (`int`) – Number of preallocated variables in the optimization task.

`Task.putnadouparam(paramname, parvalue)`
Sets the value of a named double parameter.
`Parameters`

- [in] `paramname` (`str`) – Name of a parameter.
- [in] `parvalue` (`double`) – Parameter value.

`Task.putnaintparam(paramname, parvalue)`
Sets the value of a named integer parameter.
`Parameters`

- [in] `paramname` (`str`) – Name of a parameter.
- [in] `parvalue` (`int`) – Parameter value.

`Task.putnastrparam(paramname, parvalue)`
Sets the value of a named string parameter.
`Parameters`

- [in] `paramname` (`str`) – Name of a parameter.
- [in] `parvalue` (`str`) – Parameter value.

`Task.putobjname(objname)`
Assigns the name given by `objname` to the objective function.
`Parameters`

- [in] `objname` (`str`) – Name of the objective.

`Task.putobjsense(sense)`
Sets the objective sense of the task.
`Parameters`

- [in] `sense` (*objsense*) – The objective sense of the task. The values *objsense.maximize* and *objsense.minimize* means that the problem is maximized or minimized respectively.

`Task.putparam(parname, parvalue)`
Checks if a `parname` is valid parameter name. If it is, the parameter is assigned the value specified by `parvalue`.
`Parameters`

- [in] `parname` (`str`) – Parameter name.
- [in] `parvalue` (`str`) – Parameter value.

`Task.putqcon(qcsubk, qcsubi, qcsubj, qcval)`
Replace all quadratic entries in the constraints. consider constraints on the form:

$$l_k^c \leq \frac{1}{2} \sum_{i=0}^{numvar-1} \sum_{j=0}^{numvar-1} q_{ij}^k x_i x_j + \sum_{j=0}^{numvar-1} a_{kj} x_j \leq u_k^c, \ k = 0, \ldots, m-1.$$

the function assigns values to q such that:

$$q^{\texttt{qcsubk[t]}}_{\texttt{qcsubi[t]},\texttt{qcsubj[t]}} = \texttt{qcval[t]}, \ t = 0, \ldots, \texttt{numqcnz} - 1.$$

and

$$q^{\texttt{qcsubk[t]}}_{\texttt{qcsubj[t]},\texttt{qcsubi[t]}} = \texttt{qcval[t]}, \ t = 0, \ldots, \texttt{numqcnz} - 1.$$

values not assigned are set to zero.

Please note that duplicate entries are added together.
`Parameters`

- [in] `qcsubk` (`int[]`) – Constraint subscripts for quadratic coefficients.

www.EngineeringBooksPdf.com

- [in] `qcsubi` (`int[]`) – Row subscripts for quadratic constraint matrix.
- [in] `qcsubj` (`int[]`) – Column subscripts for quadratic constraint matrix.
- [in] `qcval` (`double[]`) – Quadratic constraint coefficient values.

`Task.putqconk(k, qcsubi, qcsubj, qcval)`

Replaces all the quadratic entries in one constraint k of the form:

$$l_k^c \leq \frac{1}{2} \sum_{i=0}^{numvar-1} \sum_{j=0}^{numvar-1} q_{ij}^k x_i x_j + \sum_{j=0}^{numvar-1} a_{kj} x_j \leq u_k^c.$$

It is assumed that Q^k is symmetric, i.e. $q_{ij}^k = q_{ji}^k$,and therefore, only the values of q_{ij}^k for which $i \geq j$ should be inputted to **MOSEK**. To be precise, **MOSEK** uses the following procedure

1. $Q^k = 0$
2. for $t = 0$ to $numqcnz - 1$
3. $\quad q^k_{\texttt{qcsubi[t]},\texttt{qcsubj[t]}} = q^k_{\texttt{qcsubi[t]},\texttt{qcsubj[t]}} + \texttt{qcval[t]}$
3. $\quad q^k_{\texttt{qcsubj[t]},\texttt{qcsubi[t]}} = q^k_{\texttt{qcsubj[t]},\texttt{qcsubi[t]}} + \texttt{qcval[t]}$

Please note that:

- For large problems it is essential for the efficiency that the function *task.putmaxnumqnz* is employed to specify an appropriate `maxnumqnz`.
- Only the lower triangular part should be specified because Q^k is symmetric. Specifying values for q_{ij}^k where $i < j$ will result in an error.
- Only non-zero elements should be specified.
- The order in which the non-zero elements are specified is insignificant.
- Duplicate elements are added together. Hence, it is recommended not to specify the same element multiple times in `qcsubi`, `qcsubj`, and `qcval`.

For a code example see Section *3.5*

`Parameters`

- [in] `k` (`int`) – The constraint in which the new Q elements are inserted.
- [in] `qcsubi` (`int[]`) – Row subscripts for quadratic constraint matrix.
- [in] `qcsubj` (`int[]`) – Column subscripts for quadratic constraint matrix.
- [in] `qcval` (`double[]`) – Quadratic constraint coefficient values.

`Task.putqobj(qosubi, qosubj, qoval)`

Replaces all the quadratic terms in the objective

$$\frac{1}{2} \sum_{i=0}^{numvar-1} \sum_{j=0}^{numvar-1} q_{ij}^o x_i x_j + \sum_{j=0}^{numvar-1} c_j x_j + c^f.$$

It is assumed that Q^o is symmetric, i.e. $q_{ij}^o = q_{ji}^o$, and therefore, only the values of q_{ij}^o for which $i \geq j$ should be specified. To be precise, **MOSEK** uses the following procedure

1. $Q^o = 0$
2. for $t = 0$ to $numqonz - 1$
3. $\quad q^o_{\texttt{qosubi[t]},\texttt{qosubj[t]}} = q^o_{\texttt{qosubi[t]},\texttt{qosubj[t]}} + \texttt{qoval[t]}$
3. $\quad q^o_{\texttt{qosubj[t]},\texttt{qosubi[t]}} = q^o_{\texttt{qosubj[t]},\texttt{qosubi[t]}} + \texttt{qoval[t]}$

Please note that:

- Only the lower triangular part should be specified because Q^o is symmetric. Specifying values for q_{ij}^o where $i < j$ will result in an error.

www.EngineeringBooksPdf.com

- Only non-zero elements should be specified.
- The order in which the non-zero elements are specified is insignificant.
- Duplicate entries are added to together.

For a code example see Section *3.5*.

`Parameters`

- `[in]` `qosubi` (`int[]`) – Row subscripts for quadratic objective coefficients.
- `[in]` `qosubj` (`int[]`) – Column subscripts for quadratic objective coefficients.
- `[in]` `qoval` (`double[]`) – Quadratic objective coefficient values.

`Task.putqobjij(i, j, qoij)`

Replaces one coefficient in the quadratic term in the objective. The function performs the assignment

$$q^o_{\mathtt{ij}} = \mathtt{qoij}.$$

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 element, one at a time, is more computationally expensive than replacing all elements at once. Use *`task.putqobj`* instead whenever possible.

`Parameters`

- `[in]` `i` (`int`) – Row index for the coefficient to be replaced.
- `[in]` `j` (`int`) – Column index for the coefficient to be replaced.
- `[in]` `qoij` (`double`) – The new value for q^o_{ij}.

`Task.putskc(whichsol, skc)`

Sets the status keys for the constraints.

`Parameters`

- `[in]` `whichsol` (*`soltype`*) – Selects a solution.
- `[in]` `skc` (*`stakey`*) – Status keys for the constraints.

`Task.putskcslice(whichsol, first, last, skc)`

Sets the status keys for the constraints.

`Parameters`

- `[in]` `whichsol` (*`soltype`*) – Selects a solution.
- `[in]` `first` (`int`) – First index in the sequence.
- `[in]` `last` (`int`) – Last index plus 1 in the sequence.
- `[in]` `skc` (*`stakey`*) – Status keys for the constraints.

`Task.putskx(whichsol, skx)`

Sets the status keys for the scalar variables.

`Parameters`

- `[in]` `whichsol` (*`soltype`*) – Selects a solution.
- `[in]` `skx` (*`stakey`*) – Status keys for the variables.

`Task.putskxslice(whichsol, first, last, skx)`

Sets the status keys for the variables.

`Parameters`

- `[in]` `whichsol` (*`soltype`*) – Selects a solution.
- `[in]` `first` (`int`) – First index in the sequence.
- `[in]` `last` (`int`) – Last index plus 1 in the sequence.

www.EngineeringBooksPdf.com

- [in] skx (*stakey*) – Status keys for the variables.

Task.putslc(whichsol, slc)
Sets the s_l^c vector for a solution.
Parameters

- [in] whichsol (*soltype*) – Selects a solution.
- [in] slc (double[]) – The s_l^c vector.

Task.putslcslice(whichsol, first, last, slc)
Sets a slice of the s_l^c vector for a solution.
Parameters

- [in] whichsol (*soltype*) – Selects a solution.
- [in] first (int) – First index in the sequence.
- [in] last (int) – Last index plus 1 in the sequence.
- [in] slc (double[]) – Dual variables corresponding to the lower bounds on the constraints.

Task.putslx(whichsol, slx)
Sets the s_l^x vector for a solution.
Parameters

- [in] whichsol (*soltype*) – Selects a solution.
- [in] slx (double[]) – The s_l^x vector.

Task.putslxslice(whichsol, first, last, slx)
Sets a slice of the s_l^x vector for a solution.
Parameters

- [in] whichsol (*soltype*) – Selects a solution.
- [in] first (int) – First index in the sequence.
- [in] last (int) – Last index plus 1 in the sequence.
- [in] slx (double[]) – Dual variables corresponding to the lower bounds on the variables.

Task.putsnx(whichsol, sux)
Sets the s_n^x vector for a solution.
Parameters

- [in] whichsol (*soltype*) – Selects a solution.
- [in] sux (double[]) – The s_n^x vector.

Task.putsnxslice(whichsol, first, last, snx)
Sets a slice of the s_n^x vector for a solution.
Parameters

- [in] whichsol (*soltype*) – Selects a solution.
- [in] first (int) – First index in the sequence.
- [in] last (int) – Last index plus 1 in the sequence.
- [in] snx (double[]) – Dual variables corresponding to the conic constraints on the variables.

Task.putsolution(whichsol, skc, skx, skn, xc, xx, y, slc, suc, slx, sux, snx)
Inserts a solution into the task.
Parameters

- [in] whichsol (*soltype*) – Selects a solution.
- [in] skc (*stakey*) – Status keys for the constraints.
- [in] skx (*stakey*) – Status keys for the variables.

www.EngineeringBooksPdf.com

- [in] skn (*stakey*) – Status keys for the conic constraints.
- [in] xc (double[]) – Primal constraint solution.
- [in] xx (double[]) – Primal variable solution.
- [in] y (double[]) – Vector of dual variables corresponding to the constraints.
- [in] slc (double[]) – Dual variables corresponding to the lower bounds on the constraints.
- [in] suc (double[]) – Dual variables corresponding to the upper bounds on the constraints.
- [in] slx (double[]) – Dual variables corresponding to the lower bounds on the variables.
- [in] sux (double[]) – Dual variables corresponding to the upper bounds on the variables.
- [in] snx (double[]) – Dual variables corresponding to the conic constraints on the variables.

Task.putsolutioni(accmode, i, whichsol, sk, x, sl, su, sn)
Sets the primal and dual solution information for a single constraint or variable.
Parameters

- [in] accmode (*accmode*) – If set to *accmode.con* the solution information for a constraint is modified. Otherwise for a variable.
- [in] i (int) – Index of the constraint or variable.
- [in] whichsol (*soltype*) – Selects a solution.
- [in] sk (*stakey*) – Status key of the constraint or variable.
- [in] x (double) – Solution value of the primal constraint or variable.
- [in] sl (double) – Solution value of the dual variable associated with the lower bound.
- [in] su (double) – Solution value of the dual variable associated with the upper bound.
- [in] sn (double) – Solution value of the dual variable associated with the cone constraint.

Task.putsolutionyi(i, whichsol, y)
Inputs the dual variable of a solution.
Parameters

- [in] i (int) – Index of the dual variable.
- [in] whichsol (*soltype*) – Selects a solution.
- [in] y (double) – Solution value of the dual variable.

Task.putstrparam(param, parvalue)
Sets the value of a string parameter.
Parameters

- [in] param (*sparam*) – Which parameter.
- [in] parvalue (str) – Parameter value.

Task.putsuc(whichsol, suc)
Sets the s_u^c vector for a solution.
Parameters

- [in] whichsol (*soltype*) – Selects a solution.
- [in] suc (double[]) – The s_u^c vector.

Task.putsucslice(whichsol, first, last, suc)
Sets a slice of the s_u^c vector for a solution.
Parameters

- [in] whichsol (*soltype*) – Selects a solution.
- [in] first (int) – First index in the sequence.

www.EngineeringBooksPdf.com

- [in] `last` (`int`) – Last index plus 1 in the sequence.
- [in] `suc` (`double[]`) – Dual variables corresponding to the upper bounds on the constraints.

`Task.putsux(whichsol, sux)`
Sets the s_u^x vector for a solution.
Parameters

- [in] `whichsol` (*soltype*) – Selects a solution.
- [in] `sux` (`double[]`) – The s_u^x vector.

`Task.putsuxslice(whichsol, first, last, sux)`
Sets a slice of the s_u^x vector for a solution.
Parameters

- [in] `whichsol` (*soltype*) – Selects a solution.
- [in] `first` (`int`) – First index in the sequence.
- [in] `last` (`int`) – Last index plus 1 in the sequence.
- [in] `sux` (`double[]`) – Dual variables corresponding to the upper bounds on the variables.

`Task.puttaskname(taskname)`
Assigns the name `taskname` to the task.
Parameters

- [in] `taskname` (`str`) – Name assigned to the task.

`Task.putvarbound(j, bk, bl, bu)`
Changes the bounds for one variable.

If the a 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

- [in] `j` (`int`) – Index of the variable.
- [in] `bk` (*boundkey*) – New bound key.
- [in] `bl` (`double`) – New lower bound.
- [in] `bu` (`double`) – New upper bound.

`Task.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.
Parameters

- [in] `sub` (`int[]`) – List of variable indexes.
- [in] `bkx` (*boundkey*) – New bound keys.
- [in] `blx` (`double[]`) – New lower bound values.
- [in] `bux` (`double[]`) – New upper bound values.

`Task.putvarboundslice(first, last, bk, bl, bu)`
Changes the bounds for a slice of the variables.
Parameters

- [in] `first` (`int`) – Index of the first variable in the slice.
- [in] `last` (`int`) – Index of the last variable in the slice plus 1.
- [in] `bk` (*boundkey*) – New bound keys.
- [in] `bl` (`double[]`) – New lower bounds.
- [in] `bu` (`double[]`) – New upper bounds.

www.EngineeringBooksPdf.com

`Task.putvarname(j, name)`
Puts the name of a variable.
`Parameters`

- `[in]` `j` (`int`) – Index of the variable.
- `[in]` `name` (`str`) – The variable name.

`Task.putvartype(j, vartype)`
Sets the variable type of one variable.
`Parameters`

- `[in]` `j` (`int`) – Index of the variable.
- `[in]` `vartype` (*variabletype*) – The new variable type.

`Task.putvartypelist(subj, vartype)`
Sets the variable type for one or more variables, i.e. variable number `subj`$[k]$ is assigned the variable type `vartype`$[k]$.

If the same index is specified multiple times in `subj` only the last entry takes effect.
`Parameters`

- `[in]` `subj` (`int[]`) – A list of variable indexes for which the variable type should be changed.
- `[in]` `vartype` (*variabletype*) – A list of variable types that should be assigned to the variables specified by `subj`. See *variabletype* for the possible values of `vartype`.

`Task.putxc(whichsol, xc)`
Sets the x^c vector for a solution.
`Parameters`

- `[in]` `whichsol` (*soltype*) – Selects a solution.
- `[out]` `xc` (`double[]`) – The x^c vector.

`Task.putxcslice(whichsol, first, last, xc)`
Sets a slice of the x^c vector for a solution.
`Parameters`

- `[in]` `whichsol` (*soltype*) – Selects a solution.
- `[in]` `first` (`int`) – First index in the sequence.
- `[in]` `last` (`int`) – Last index plus 1 in the sequence.
- `[in]` `xc` (`double[]`) – Primal constraint solution.

`Task.putxx(whichsol, xx)`
Sets the x^x vector for a solution.
`Parameters`

- `[in]` `whichsol` (*soltype*) – Selects a solution.
- `[in]` `xx` (`double[]`) – The x^x vector.

`Task.putxxslice(whichsol, first, last, xx)`
Obtains a slice of the x^x vector for a solution.
`Parameters`

- `[in]` `whichsol` (*soltype*) – Selects a solution.
- `[in]` `first` (`int`) – First index in the sequence.
- `[in]` `last` (`int`) – Last index plus 1 in the sequence.
- `[in]` `xx` (`double[]`) – Primal variable solution.

`Task.puty(whichsol, y)`
Sets the y vector for a solution.
`Parameters`

www.EngineeringBooksPdf.com

- [in] whichsol (*soltype*) – Selects a solution.
- [in] y (double[]) – The y vector.

Task.putyslice(whichsol, first, last, y)
Sets a slice of the y vector for a solution.
Parameters

- [in] whichsol (*soltype*) – Selects a solution.
- [in] first (int) – First index in the sequence.
- [in] last (int) – Last index plus 1 in the sequence.
- [in] y (double[]) – Vector of dual variables corresponding to the constraints.

Task.readdata(filename)
Reads an optimization problem and associated data from a file.
Parameters

- [in] filename (str) – Data is read from the file filename.

Task.readdataformat(filename, format, compress)
Reads an optimization problem and associated data from a file.
Parameters

- [in] filename (str) – Data is read from the file filename.
- [in] format (*dataformat*) – File data format.
- [in] compress (*compresstype*) – File compression type.

Task.readparamfile(filename)
Reads a parameter file.
Parameters

- [in] filename (str) – 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*.

Task.readsolution(whichsol, filename)
Reads a solution file and inserts the solution into the solution whichsol.
Parameters

- [in] whichsol (*soltype*) – Selects a solution.
- [in] filename (str) – A valid file name.

Task.readsummary(whichstream)
Prints a short summary of last file that was read.
Parameters

- [in] whichstream (*streamtype*) – Index of the stream.

Task.readtask(filename)
Load task data from a file, replacing any data that already is in the task object. All problem data are resorted, but if the file contains solutions, the solution status after loading a file is still unknown, even if it was optimal or otherwise well-defined when the file was dumped.

See section *17.6* for a description of the Task format.
Parameters

- [in] filename (str) – Input file name.

Task.removebarvars(subset)
The function removes a subset of the symmetric matrix from the optimization task. This implies that the existing symmetric matrix are renumbered, for instance if constraint 5 is removed then constraint 6 becomes constraint 5 and so forth.
Parameters

- [in] subset (int[]) – Indexes of symmetric matrix which should be removed.

www.EngineeringBooksPdf.com

`Task.removecones(subset)`
Removes a number conic constraint from the problem. In general, it is much more efficient to remove a cone with a high index than a low index.
`Parameters`

- `[in]` `subset` (`int[]`) – Indexes of cones which should be removed.

`Task.removecons(subset)`
The function removes a subset of the constraints from the optimization task. This implies that the existing constraints are renumbered, for instance if constraint 5 is removed then constraint 6 becomes constraint 5 and so forth.
`Parameters`

- `[in]` `subset` (`int[]`) – Indexes of constraints which should be removed.

`Task.removevars(subset)`
The function removes a subset of the variables from the optimization task. This implies that the existing variables are renumbered, for instance if constraint 5 is removed then constraint 6 becomes constraint 5 and so forth.
`Parameters`

- `[in]` `subset` (`int[]`) – Indexes of variables which should be removed.

`Task.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 its only function is to give a hint of the amount of data to preallocate for efficiency reasons.

Please note that the procedure is **destructive** in the sense that all existing data stored in the task is destroyed.
`Parameters`

- `[in]` `maxnumcon` (`int`) – New maximum number of constraints.
- `[in]` `maxnumvar` (`int`) – New maximum number of variables.
- `[in]` `maxnumcone` (`int`) – New maximum number of cones.
- `[in]` `maxnumanz` (`long`) – New maximum number of non-zeros in A.
- `[in]` `maxnumqnz` (`long`) – New maximum number of non-zeros in all Q matrices.

`Task.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`

- `[in]` `whichstream` (*streamtype*) – Index of the stream.

`Task.set_Progress(callback)`
Recieve callbacks about current status of the solver during optimization.

For example:

Returns non-zero to request that the solver terminates.
`Parameters`

- `[in]` `callback` (`none`) – The callback function.

`Task.set_Stream(whichstream, callback)`
Directs all output from a task stream to a callback function.
`Parameters`

- `[in]` `whichstream` (*streamtype*) – Index of the stream.
- `[in]` `callback` (`none`) – The callback function.

www.EngineeringBooksPdf.com

`Task.setdefaults()`
Resets all the parameters to their default values.

`isdef` = `Task.solutiondef(whichsol)`
Checks whether a solution is defined.
`Parameters`

- `[in]` `whichsol` (*soltype*) – Selects a solution.

`Return`

- `isdef` (`int`) – Is non-zero if the requested solution is defined.

`Task.solutionsummary(whichstream)`
Prints a short summary of the current solutions.
`Parameters`

- `[in]` `whichstream` (*streamtype*) – Index of the stream.

`numnz` = `Task.solvewithbasis(transp, numnz, sub, val)`
If a basic solution is available, then exactly `numcon` basis variables are defined. These `numcon` basis variables are denoted the basis. Associated with the basis is a basis matrix denoted B. This function solves either the linear equation system

$$B\overline{X} = b \tag{16.3}$$

or the system

$$B^T\overline{X} = b \tag{16.4}$$

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 kth basis variable is variable x_j it implies that

$$B_{i,k} = A_{i,j},\ i = 0, \ldots, numcon - 1.$$

Otherwise if the kth basis variable is variable x_j^c it implies that

$$B_{i,k} = \begin{cases} -1, & i = j, \\ 0, & i \neq j. \end{cases}$$

Given the knowledge of how B is constructed it is possible to interpret the solution $\overline{X}$ correctly.

Please note that this function exploits the sparsity in the vector b to speed up the computations.
`Parameters`

- `[in]` `transp` (`int`) – If this argument is non-zero, then (16.4) is solved. Otherwise the system (16.3) is solved.
- `[io]` `numnz` (`int`) – As input it is the number of non-zeros in b. As output it is the number of non-zeros in $\overline{X}$.
- `[io]` `sub` (`int[]`) – As input it contains the positions of the non-zeros in b, i.e.

  $$b[\texttt{sub}[k]] \neq 0, \quad k = 0, \ldots, numnz[0] - 1.$$

  As output it contains the positions of the non-zeros in $\overline{X}$. It is important that `sub` has room for `numcon` elements.
- `[io]` `val` (`double[]`) – As input it is the vector b. Although the positions of the non-zero elements are specified in `sub` it is required that `val`$[i] = 0$ if $b[i] = 0$. As output `val` is the vector $\overline{X}$.

  Please note that `val` is a dense vector — not a packed sparse vector. This implies that `val` has room for `numcon` elements.

www.EngineeringBooksPdf.com

Return

- **numnz** (int) – As input it is the number of non-zeros in b. As output it is the number of non-zeros in $\overline{X}$.

conetype = Task.strtoconetype(str)
Obtains cone type code corresponding to a cone type string.
Parameters

- [in] **str** (str) – String corresponding to the cone type code codetype.

Return

- **conetype** (int) – The cone type corresponding to the string str.

sk = Task.strtosk(str)
Obtains the status key corresponding to an explanatory string.
Parameters

- [in] **str** (str) – Status key string.

Return

- **sk** (int) – Status key corresponding to the string.

Task.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 as a constraint, if any. 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.

Task.updatesolutioninfo(whichsol)
Update the information items related to the solution.
Parameters

- [in] **whichsol** (*soltype*) – Selects a solution.

Task.writeSC(scfilename, taskfilename)
Write problem to an SCopt file and a normal problem file.
Parameters

- [in] **scfilename** (str) – Name of SCopt terms file.
- [in] **taskfilename** (str) – Name of problem file.

Task.writedata(filename)
Writes problem data associated with the optimization task to a file in one of the supported formats. See Section *17* for the complete list.

By default the data file format is determined by the file name extension. This behaviour can be overridden by setting the *iparam.write_data_format* parameter.

MOSEK is able to read and write files in a compressed format (gzip). To write in the compressed format append the extension .gz. E.g to write a gzip compressed MPS file use the extension mps.gz.

Please note that MPS, LP and OPF files require all variables to have unique names. If a task contains no names, it is possible to write the file with automatically generated anonymous names by setting the *iparam.write_generic_names* parameter to *onoffkey.on*.

www.EngineeringBooksPdf.com

Please note that if a general nonlinear function appears in the problem then such function *cannot* be written to file and **MOSEK** will issue a warning.

Parameters

- [in] `filename` (`str`) – 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*.

`Task.writejsonsol(filename)`

Saves the current solutions and solver information items in a JSON file.

Parameters

- [in] `filename` (`str`) – A valid file name.

`Task.writeparamfile(filename)`

Writes all the parameters to a parameter file.

Parameters

- [in] `filename` (`str`) – The name of parameter file.

`Task.writesolution(whichsol, filename)`

Saves the current basic, interior-point, or integer solution to a file.

Parameters

- [in] `whichsol` (*soltype*) – Selects a solution.
- [in] `filename` (`str`) – A valid file name.

`Task.writetask(filename)`

Write a binary dump of the task data. This format saves all problem data, but not callback-functions and general non-linear terms.

See section *17.6* for a description of the Task format.

Parameters

- [in] `filename` (`str`) – Output file name.

`Task.writetasksolverresult_file(filename)`

Internal

Parameters

- [in] `filename` (`str`) – A valid file name.

16.10 Exceptions

`Exception`

Base exception class for all **MOSEK** exceptions.

Implements

Exception

`Error`

Exception class used for all error response codes from **MOSEK**.

Implements

Exception

`Warning`

Exception class used for all warning response codes from **MOSEK**.

Implements

Exception

www.EngineeringBooksPdf.com

www.EngineeringBooksPdf.com

CHAPTER

SEVENTEEN

SUPPORTED FILE FORMATS

MOSEK supports a range of problem and solution formats listed in Table 17.1 and Table 17.2. 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

See Table 17.1.

Table 17.1: List of supported file formats for optimization problems.

Format Type	Ext.	Binary/Text	LP	QP	CQO	SDP
LP	lp	plain text	X	X		
MPS	mps	plain text	X	X		
OPF	opf	plain text	X	X	X	
CBF	cbf	plain text	X		X	X
Osil	xml	xml text	X	X		
Task format	task	binary	X	X	X	X
Jtask format	jtask	text	X	X	X	X

Solution formats

See Table 17.2.

Table 17.2: 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 compression of files. Problem files with an additional `.gz` extension are assumed to be compressed when read, and are automatically compressed when written. For example, a file called

www.EngineeringBooksPdf.com

```
problem.mps.gz
```

will be considered as a GZIP compressed MPS file.

17.1 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 on the form

$$\begin{array}{lccccc} \text{minimize/maximize} & & & c^T x + \frac{1}{2} q^o(x) & & \\ \text{subject to} & l^c & \leq & Ax + \frac{1}{2} q(x) & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x, \\ & & & x_{\mathcal{J}} \text{ integer}, & & \end{array}$$

where

- $x \in \mathbb{R}^n$ is the vector of decision variables.
- $c \in \mathbb{R}^n$ is the linear term in the objective.
- $q^o :\in \mathbb{R}^n \to \mathbb{R}$ is the quadratic term in the objective where

$$q^o(x) = x^T Q^o x$$

and it is assumed that

$$Q^o = (Q^o)^T.$$

- $A \in \mathbb{R}^{m \times n}$ is the constraint matrix.
- $l^c \in \mathbb{R}^m$ is the lower limit on the activity for the constraints.
- $u^c \in \mathbb{R}^m$ is the upper limit on the activity for the constraints.
- $l^x \in \mathbb{R}^n$ is the lower limit on the activity for the variables.
- $u^x \in \mathbb{R}^n$ is the upper limit on the activity for the variables.
- $q : \mathbb{R}^n \to \mathbb{R}$ is a vector of quadratic functions. Hence,

$$q_i(x) = x^T Q^i x$$

where it is assumed that

$$Q^i = (Q^i)^T.$$

- $\mathcal{J} \subseteq \{1, 2, \ldots, n\}$ is an index set of the integer constrained variables.

17.1.1 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

www.EngineeringBooksPdf.com

```
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 (`[ ]`) 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
```


www.EngineeringBooksPdf.com

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, but **MOSEK** supports defining ranged constraints by using double-colon (`::`) instead of a single-colon (`:`) after the constraint name, i.e.

$$-5 \leq x_1 + x_2 \leq 5 \tag{17.1}$$

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 (17.1) 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:

www.EngineeringBooksPdf.com

```
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
```

17.1.2 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
```

17.1.3 LP Format peculiarities

Comments

Anything on a line after a \ is ignored and is treated as a comment.

www.EngineeringBooksPdf.com

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.

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 loose their uniqueness and change the problem.

To get around some of the inconveniences converting from other problem formats, **MOSEK** allows lines to contain 1024 characters and names may have any length (shorter than the 1024 characters).

Internally in **MOSEK** names may contain any (printable) character, many of which cannot be used in LP names. Setting the parameters

- *iparam.read_lp_quoted_names* and
- *iparam.write_lp_quoted_names*

www.EngineeringBooksPdf.com

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.

17.1.4 The strict LP format

The LP format is not a formal standard and different vendors have slightly different interpretations of the LP format. To make **MOSEK**'s definition of the LP format more compatible with the definitions of other vendors, use the parameter setting

- *iparam.write_lp_strict_format* = *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 use the parameter setting

- *iparam.write_generic_names* = *onoffkey.on*

which will cause all names to be renamed systematically in the output file.

17.1.5 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*
- *iparam.write_lp_terms_per_line*

The first parameter sets the maximum number of characters on a single line. The default value is 80 corresponding roughly to the width of a standard text document.

The second parameter sets the maximum number of terms per line; a term means a sign, a coefficient, and a name (for example `+ 42 elephants`). The default value is 0, meaning that there is no maximum.

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).

17.2 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]*.

17.2.1 MPS File Structure

The version of the MPS format supported by **MOSEK** allows specification of an optimization problem of the form

$$\begin{array}{rcccl} l^c & \leq & Ax + q(x) & \leq & u^c, \\ l^x & \leq & x & \leq & u^x, \\ & & x \in \mathcal{K}, & & \\ & & x_{\mathcal{J}} \ \text{integer}, & & \end{array} \tag{17.2}$$

where

www.EngineeringBooksPdf.com

- $x \in \mathbb{R}^n$ 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.

- $\mathcal{K}$ is a convex cone.
- $\mathcal{J} \subseteq \{1, 2, \ldots, n\}$ is an index set of the integer-constrained variables.

An MPS file with one row and one column can be illustrated like this:

```
*        1         2         3         4         5         6
*2345678901234567890123456789012345678901234567890123456789012345678901234567890
NAME          [name]
OBJSENSE
[objsense]
OBJNAME
[objname]
ROWS
?  [cname1]
COLUMNS
[vname1]  [cname1]    [value1]     [vname3]  [value2]
RHS
[name]    [cname1]    [value1]     [cname2]  [value2]
RANGES
[name]    [cname1]    [value1]     [cname2]  [value2]
QSECTION      [cname1]
[vname1]  [vname2]    [value1]     [vname3]  [value2]
QMATRIX
[vname1]  [vname2]    [value1]
QUADOBJ
[vname1]  [vname2]    [value1]
QCMATRIX      [cname1]
[vname1]  [vname2]    [value1]
BOUNDS
?? [name]    [vname1]    [value1]
CSECTION      [kname1]    [value1]     [ktype]
[vname1]
ENDATA
```

Here the names in capitals are keywords of the MPS format and names in brackets are custom defined names or values. A couple of notes on the structure:

- Fields: All items surrounded by brackets appear in *fields*. The fields named "`valueN`" are numerical values. Hence, they must have the format

```
  [+|-]XXXXXXX.XXXXXX[[e|E][+|-]XXX]


where
```


www.EngineeringBooksPdf.com

```
.. code-block:: text

   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 Section *17.2.9* for details.

Linear example `lo1.mps`

A concrete example of a MPS file is presented below:

```
* File: lo1.mps
NAME          lo1
OBJSENSE
    MAX
ROWS
 N  obj
 E  c1
 G  c2
 L  c3
COLUMNS
    x1        obj       3
    x1        c1        3
    x1        c2        2
    x2        obj       1
    x2        c1        1
    x2        c2        1
    x2        c3        2
    x3        obj       5
    x3        c1        2
    x3        c2        3
    x4        obj       1
    x4        c2        1
    x4        c3        3
RHS
    rhs       c1        30
    rhs       c2        15
    rhs       c3        25
RANGES
BOUNDS
 UP bound     x2        10
ENDATA
```

Subsequently each individual section in the MPS format is discussed.

Section `NAME`

In this section a name (`[name]`) is assigned to the problem.

www.EngineeringBooksPdf.com

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. The OBJNAME section contains one line at most which has the form

```
objname
```

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 an unique name denoted by [cname1]. Please note that [cname1] starts in position 5 and the field can be at most 8 characters wide. An initial key ? must be present to specify the type of the constraint. The key can have the values E, G, L, or N with the following interpretation:

Constraint type	l_i^c	u_i^c
E	finite	l_i^c
G	finite	∞
L	$-\infty$	finite
N	$-\infty$	∞

In the MPS format an objective vector is not specified explicitly, but one of the constraints having the key N will be used as the objective vector c . In general, if multiple N type constraints are specified, then the first will be used as the objective vector c.

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:

www.EngineeringBooksPdf.com

Field	Starting Position	Max Width	required	Description
[vname1]	5	8	Yes	Variable name
[cname1]	15	8	Yes	Constraint name
[value1]	25	12	Yes	Numerical value
[cname2]	40	8	No	Constraint name
[value2]	50	12	No	Numerical value

Hence, a record specifies one or two elements a_{ij} of A using the principle that [vname1] and [cname1] determines j and i respectively. Please note that [cname1] must be a constraint name specified in the ROWS section. Finally, [value1] denotes the numerical value of a_{ij} . Another optional element is specified by [cname2], and [value2] for the variable specified by [vname1]. Some important comments are:

- All elements belonging to one variable must be grouped together.
- Zero elements of A should not be specified.
- At least one element for each variable should be specified.

RHS (optional)

A record in this section has the format

```
[name]    [cname1]    [value1]    [cname2]  [value2]
```

where the requirements for each field are as follows:

Field	Starting Position	Max Width	required	Description
[name]	5	8	Yes	Name of the RHS vector
[cname1]	15	8	Yes	Constraint name
[value1]	25	12	Yes	Numerical value
[cname2]	40	8	No	Constraint name
[value2]	50	12	No	Numerical value

The interpretation of a record is that [name] is the name of the RHS vector to be specified. In general, several vectors can be specified. [cname1] denotes a constraint name previously specified in the ROWS section. Now, assume that this name has been assigned to the i th constraint and v_1 denotes the value specified by [value1], then the interpretation of v_1 is:

Constraint	l_i^c	u_i^c
type		
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

www.EngineeringBooksPdf.com

The records in this section are used to modify the bound vectors for the constraints, i.e. the values in l^c and u^c . A record has the following interpretation: `[name]` is the name of the `RANGE` vector and `[cname1]` is a valid constraint name. Assume that `[cname1]` is assigned to the i th constraint and let v_1 be the value specified by `[value1]`, then a record has the interpretation:

Constraint type	Sign of v_1	l_i^c	u_i^c
E	$-$	$u_i^c + v_1$	
E	$+$		$l_i^c + v_1$
G	$-$ or $+$	$l_i^c + \|v_1\|$	
L	$-$ or $+$	$u_i^c - \|v_1\|$	
N			

`QSECTION` (optional)

Within the `QSECTION` the label `[cname1]` must be a constraint name previously specified in the `ROWS` section. The label `[cname1]` denotes the constraint to which the quadratic term belongs. A record in the `QSECTION` has the form

```
[vname1]   [vname2]     [value1]      [vname3]   [value2]
```

where the requirements for each field are:

Field	Starting Position	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

$$\begin{array}{lcl} \text{minimize} & -x_2 + \frac{1}{2}(2x_1^2 - 2x_1x_3 + 0.2x_2^2 + 2x_3^2) & \\ \text{subject to} & x_1 + x_2 + x_3 \geq 1, & \\ & x \geq 0 & \end{array}$$

has the following MPS file representation

```
* File: qo1.mps
NAME          qo1
ROWS
N  obj
G  c1
COLUMNS
x1        c1        1.0
x2        obj       -1.0
x2        c1        1.0
x3        c1        1.0
RHS
rhs       c1        1.0
QSECTION      obj
x1        x1        2.0
x1        x3        -1.0
x2        x2        0.2
x3        x3        2.0
ENDATA
```

Regarding the `QSECTION`s please note that:

www.EngineeringBooksPdf.com

- Only one `QSECTION` is allowed for each constraint.
- The `QSECTION`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.
- 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` It stores all the nonzeros coefficients, withouot taking advantage of the symmetry of the Q matrix.
- `QUADOBJ` It only store 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 apper for each off-diagonal coefficient if using a `QMATRIX` section, while only one entry is required in a `QUADOBJ` section. The quadratic part of the objective function will be evaluated as $1/2x^TQx$.

The example

$$\begin{array}{lcl} \text{minimize} & -x_2 + \frac{1}{2}(2x_1^2 - 2x_1x_3 + 0.2x_2^2 + 2x_3^2) & \\ \text{subject to} & x_1 + x_2 + x_3 \geq 1, & \\ & x \geq 0 & \end{array}$$

has the following MPS file representation using `QMATRIX`

```
* File: qo1_matrix.mps
NAME          qo1_qmatrix
ROWS
 N  obj
 G  c1
COLUMNS
    x1        c1        1.0
    x2        obj       -1.0
    x2        c1        1.0
    x3        c1        1.0
RHS
    rhs       c1        1.0
QMATRIX
    x1        x1        2.0
    x1        x3        -1.0
    x3        x1        -1.0
    x2        x2        0.2
    x3        x3        2.0
ENDATA
```


www.EngineeringBooksPdf.com

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.

17.2.2 QCMATRIX (optional)

A QCMATRIX section allows to specify the quadratic part of a given constraints. 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

$$\begin{array}{lccc} \text{minimize} & x_2 & & \\ \text{subject to} & x_1+x_2+x_3 & \geq & 1, \\ & \frac{1}{2}(-2x_1x_3+0.2x_2^2+2x_3^2) & \leq & 10, \\ & x \geq 0 & & \end{array}$$

has the following MPS file representation

```
* File: qo1.mps
NAME          qo1
ROWS
 N  obj
 G  c1
 L  q1
COLUMNS
```


www.EngineeringBooksPdf.com

```
    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.
- A `QCMATRIX` does not exploit the symmetry of Q: an off-diagonal entry (i, j) should apperas twice.

17.2.3 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 which bounds are modified by the record. `??` and `[value1]` are used to modify the bound vectors according to the following table:

`??`	l_j^x	u_j^x	Made integer (added to $\mathcal{J}$)
`FR`	$-\infty$	∞	No
`FX`	v_1	v_1	No
`LO`	v_1	unchanged	No
`MI`	$-\infty$	unchanged	No
`PL`	unchanged	∞	No
`UP`	unchanged	v_1	No
`BV`	0	1	Yes
`LI`	$\lceil v_1 \rceil$	unchanged	Yes
`UI`	unchanged	$\lfloor v_1 \rfloor$	Yes

v_ 1 is the value specified by `[value1]`.

www.EngineeringBooksPdf.com

17.2.4 CSECTION (optional)

The purpose of the CSECTION is to specify the constraint

$$x \in \mathcal{K}.$$

in (17.2). 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:

$$\mathcal{K}_t = \left\{ x \in \mathbb{R}^{n^t} \right\}.$$

- Quadratic cone:

$$\mathcal{K}_t = \left\{ x \in \mathbb{R}^{n^t} : x_1 \geq \sqrt{\sum_{j=2}^{n^t} x_j^2} \right\}. \tag{17.3}$$

- Rotated quadratic cone:

$$\mathcal{K}_t = \left\{ x \in \mathbb{R}^{n^t} : 2x_1 x_2 \geq \sum_{j=3}^{n^t} x_j^2, \quad x_1, x_2 \geq 0 \right\}. \tag{17.4}$$

In general, only quadratic and rotated quadratic cones are specified in the MPS file whereas membership of the $\mathbb{R}$ set is not. If a variable is not a member of any other cone then it is assumed to be a member of an $\mathbb{R}$ cone.

Next, let us study an example. Assume that the quadratic cone

$$x_4 \geq \sqrt{x_5^2 + x_8^2}$$

and the rotated quadratic cone

$$x_3 x_7 \geq x_1^2 + x_0^2, \quad x_3, x_7 \geq 0,$$

should be specified in the MPS file. One CSECTION is required for each cone and they are specified as follows:

www.EngineeringBooksPdf.com

```
*        1         2         3         4         5         6
*2345678901234567890123456789012345678901234567890123456789012345678901234567890
CSECTION       konea        0.0          QUAD
x4
x5
x8
CSECTION       koneb        0.0          RQUAD
x7
x3
x1
x0
```

This first `CSECTION` specifies the cone (17.3) which is given the name `konea`. This is a quadratic cone which is specified by the keyword `QUAD` in the `CSECTION` header. The 0.0 value in the `CSECTION` header is not used by the `QUAD` cone.

The second `CSECTION` specifies the rotated quadratic cone (17.4). Please note the keyword `RQUAD` in the `CSECTION` which is used to specify that the cone is a rotated quadratic cone instead of a quadratic cone. The 0.0 value in the `CSECTION` header is not used by the `RQUAD` cone.

In general, a `CSECTION` header has the format

```
CSECTION       [kname1]     [value1]     [ktype]
```

where the requirement for each field are as follows:

Field	Starting Position	Max Width	Required	Description
[kname1]	5	8	Yes	Name of the cone
[value1]	15	12	No	Cone parameter
[ktype]	25		Yes	Type of the cone.

The possible cone type keys are:

Cone type key	Members	Interpretation.
QUAD	≤ 1	Quadratic cone i.e. (17.3).
RQUAD	≤ 2	Rotated quadratic cone i.e. (17.4).

Please note that a quadratic cone must have at least one member whereas a rotated quadratic cone must have at least two members. A record in the `CSECTION` has the format

```
[vname1]
```

where the requirements for each field are

Field	Starting Position	Max Width	required	Description
[vname1]	2	8	Yes	A valid variable name

The most important restriction with respect to the `CSECTION` is that a variable must occur in only one `CSECTION`.

17.2.5 ENDATA

This keyword denotes the end of the MPS file.

17.2.6 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:

www.EngineeringBooksPdf.com

```
COLUMNS
x1        obj       -10.0          c1        0.7
x1        c2        0.5            c3        1.0
x1        c4        0.1
* Start of integer-constrained variables.
MARK000   'MARKER'                 'INTORG'
x2        obj       -9.0           c1        1.0
x2        c2        0.8333333333   c3        0.66666667
x2        c4        0.25
x3        obj       1.0            c6        2.0
MARK001   'MARKER'                 'INTEND'
```

- End of integer-constrained variables.

Please note that special marker lines are used to indicate the start and the end of the integer variables. Furthermore be aware of the following

- IMPORTANT: All variables between the markers are assigned a default lower bound of 0 and a default upper bound of 1. **This may not be what is intended.** If it is not intended, the correct bounds should be defined in the `BOUNDS` section of the MPS formatted file.
- **MOSEK** ignores field 1, i.e. `MARK0001` and `MARK001`, however, other optimization systems require them.
- Field 2, i.e. `MARKER`, must be specified including the single quotes. This implies that no row can be assigned the name `MARKER`.
- Field 3 is ignored and should be left blank.
- Field 4, i.e. `INTORG` and `INTEND`, must be specified.
- It is possible to specify several such integer marker sections within the `COLUMNS` section.

17.2.7 General Limitations

- An MPS file should be an ASCII file.

17.2.8 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.

17.2.9 The Free MPS Format

MOSEK supports a free format variation of the MPS format. The free format is similar to the MPS file format but less restrictive, e.g. it allows longer names. However, it also presents two main limitations:

- A name must not contain any blanks.
- By default a line in the MPS file must not contain more than 1024 characters. However, by modifying the parameter *iparam.read_mps_width* an arbitrary large line width will be accepted.

To use the free MPS format instead of the default MPS format the **MOSEK** parameter *iparam.read_mps_format* should be changed.

www.EngineeringBooksPdf.com

17.3 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.

17.3.1 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 in quotes [/tag]
```

Unnamed arguments are identified by their order, while named arguments may appear in any order, but never before an unnamed argument. The `value` can be a quoted, single-quoted or double-quoted text string, i.e.

```
[tag 'value']     single-quoted value [/tag]
[tag arg='value'] single-quoted value [/tag]
[tag "value"]     double-quoted value [/tag]
[tag arg="value"] double-quoted value [/tag]
```


www.EngineeringBooksPdf.com

Sections

The recognized tags are

[comment]

A comment section. This can contain *almost* any text: Between single quotes (') or double quotes (") any text may appear. Outside quotes the markup characters ([and]) must be prefixed by backslashes. Both single and double quotes may appear alone or inside a pair of quotes if it is prefixed by a backslash.

[objective]

The objective function: This accepts one or two parameters, where the first one (in the above example `min`) is either `min` or `max` (regardless of case) and defines the objective sense, and the second one (above `myobj`), if present, is the objective name. The section may contain linear and quadratic expressions. If several objectives are specified, all but the last are ignored.

[constraints]

This does not directly contain any data, but may contain the subsection `con` defining a linear constraint.

`[con]` defines a single constraint; if an argument is present (`[con NAME]`) this is used as the name of the constraint, otherwise it is given a null-name. The section contains a constraint definition written as linear and quadratic expressions with a lower bound, an upper bound, with both or with an equality. Examples:

```
[constraints]
[con 'con1'] 0 <= x + y       [/con]
[con 'con2'] 0 >= x + y       [/con]
[con 'con3'] 0 <= x + y <= 10 [/con]
[con 'con4']      x + y  = 10 [/con]
[/constraints]
```

Constraint names are unique. If a constraint is specified which has the same name as a previously defined constraint, the new constraint replaces the existing one.

[bounds]

This does not directly contain any data, but may contain the subsections `b` (linear bounds on variables) and `cone` (quadratic cone).

`[b]`. Bound definition on one or several variables separated by comma (`,`). An upper or lower bound on a variable replaces any earlier defined bound on that variable. If only one bound (upper or lower) is given only this bound is replaced. This means that upper and lower bounds can be specified separately. So the OPF bound definition:

```
[b]  x,y >= -10  [/b]
[b]  x,y <= 10   [/b]
```

results in the bound $-10 \leq x, y \leq 10$.

`[cone]`. currently supports the *quadratic cone* and the *rotated quadratic cone*.

A conic constraint is defined as a set of variables which belong to a single unique cone.

www.EngineeringBooksPdf.com

- A quadratic cone of n variables $x_1, \ldots, x_n$ defines a constraint of the form

$$x_1^2 > \sum_{i=2}^{n} x_i^2.$$

- A rotated quadratic cone of n variables $x_1, \ldots, x_n$ defines a constraint of the form

$$x_1 x_2 > \sum_{i=3}^{n} x_i^2.$$

A `[bounds]`-section example:

```
[bounds]
[b]  0 <= x,y <= 10  [/b] # ranged bound
[b] 10 >= x,y >=  0  [/b] # ranged bound
[b]  0 <= x,y <= inf [/b] # using inf
[b]       x,y free   [/b] # free variables
# Let (x,y,z,w) belong to the cone K
[cone quad]  x,y,z,w  [/cone] # quadratic cone
[cone rquad] x,y,z,w  [/cone] # rotated quadratic cone
[/bounds]
```

By default all variables are free.

[variables]

This defines an ordering of variables as they should appear in the problem. This is simply a space-separated list of variable names.

[integer]

This contains a space-separated list of variables and defines the constraint that the listed variables must be integer values.

[hints]

This may contain only non-essential data; for example estimates of the number of variables, constraints and non-zeros. Placed before all other sections containing data this may reduce the time spent reading the file.

In the `hints` section, any subsection which is not recognized by **MOSEK** is simply ignored. In this section a hint in a subsection is defined as follows:

```
[hint ITEM] value [/hint]
```

where `ITEM` may be replaced by `numvar` (number of variables), `numcon` (number of linear/quadratic constraints), `numanz` (number of linear non-zeros in constraints) and `numqnz` (number of quadratic non-zeros in constraints).

[solutions]

This section can contain a set of full or partial solutions to a problem. Each solution must be specified using a `[solution]`-section, i.e.

www.EngineeringBooksPdf.com

```
[solutions]
[solution]...[/solution] #solution 1
[solution]...[/solution] #solution 2
#other solutions....
[solution]...[/solution] #solution n
[/solutions]
```

Note that a `[solution]`-section must be always specified inside a `[solutions]`-section. 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.

www.EngineeringBooksPdf.com

  - `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.

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]+)?
```


www.EngineeringBooksPdf.com

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"
```

17.3.2 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]
```

17.3.3 Writing OPF Files from MOSEK

To write an OPF file set the parameter *iparam.write_data_format* to *dataformat.op* as this ensures that `OPF` format is used.

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.

17.3.4 Examples

This section contains a set of small examples written in OPF and describing how to formulate linear, quadratic and conic problems.

www.EngineeringBooksPdf.com

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 17.1.

Listing 17.1: 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]

[bounds]
  [b] 0 <= * [/b]
  [b] 0 <= x2 <= 10 [/b]
[/bounds]
```

Quadratic Example `qo1.opf`

An example of a quadratic optimization problem is

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

This can be formulated in `opf` as shown below.

www.EngineeringBooksPdf.com

Listing 17.2: 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:

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

Please note that the type of the cones is defined by the parameter to `[cone ...]`; the content of the `cone`-section is the names of variables that belong to the cone. The resulting OPF file is in Listing 17.3.

Listing 17.3: 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]
```


www.EngineeringBooksPdf.com

```
[objective minimize 'obj']
   x4 + x5 + x6
[/objective]

[constraints]
  [con 'c1']  x1 + x2 + 2e+00 x3 = 1e+00 [/con]
[/constraints]

[bounds]
  # We let all variables default to the positive orthant
  [b] 0 <= * [/b]

  # ...and change those that differ from the default
  [b] x4,x5,x6 free [/b]

  # Define quadratic cone: x4 >= sqrt( x1^2 + x2^2 )
  [cone quad 'k1'] x4, x1, x2 [/cone]

  # Define rotated quadratic cone: 2 x5 x6 >= x3^2
  [cone rquad 'k2'] x5, x6, x3 [/cone]
[/bounds]
```

Mixed Integer Example `milo1.opf`

Consider the mixed integer problem:

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

This can be implemented in `OPF` with the file in Listing 17.4.

Listing 17.4: Example of an OPF file for a mixed-integer linear problem.

```
[comment]
  The milo1 example in OPF format
[/comment]

[hints]
  [hint NUMVAR] 2 [/hint]
  [hint NUMCON] 2 [/hint]
  [hint NUMANZ] 4 [/hint]
[/hints]

[variables disallow_new_variables]
  x1 x2
[/variables]

[objective maximize 'obj']
   x1 + 6.4e-1 x2
[/objective]

[constraints]
  [con 'c1'] 5e+1 x1 + 3.1e+1 x2 <= 2.5e+2 [/con]
  [con 'c2'] -4 <= 3 x1 - 2 x2 [/con]
[/constraints]

[bounds]
  [b] 0 <= * [/b]
[/bounds]
```


www.EngineeringBooksPdf.com

```
[integer]
  x1 x2
[/integer]
```

17.4 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.

17.4.1 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:

$$\begin{array}{lll} \min / \max & g^{obj} & \\ & g_i \in \mathcal{K}_i, & i \in \mathcal{I}, \\ \text{s.t.} & G_i \in \mathcal{K}_i, & i \in \mathcal{I}^{PSD}, \\ & x_j \in \mathcal{K}_j, & j \in \mathcal{J}, \\ & \overline{X}_j \in \mathcal{K}_j, & j \in \mathcal{J}^{PSD}. \end{array} \tag{17.5}$$

- **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

www.EngineeringBooksPdf.com

$$\{x \in \mathbb{R}^n \mid x_j \leq 0 \text{ for } j = 1, \ldots, n\}, \text{ for } n \geq 1.$$

- **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

$$\left\{ \begin{pmatrix} p \\ q \\ x \end{pmatrix} \in \mathbb{R} \times \mathbb{R} \times \mathbb{R}^{n-2}, \; 2pq \geq x^T x, \; p \geq 0, \; q \geq 0 \right\}, \text{ for } n \geq 3.$$

17.4.2 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.

www.EngineeringBooksPdf.com

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.

17.4.3 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

$$x \in \mathcal{K}_1^{n_1} \times \mathcal{K}_2^{n_2} \times \cdots \times \mathcal{K}_k^{n_k}$$

www.EngineeringBooksPdf.com

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 17.3. 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 17.3.

- **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

www.EngineeringBooksPdf.com

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 17.3: 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

17.4.4 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.

www.EngineeringBooksPdf.com

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 17.3), 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:

www.EngineeringBooksPdf.com

```
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 17.3), 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
```


www.EngineeringBooksPdf.com

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
```


www.EngineeringBooksPdf.com

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.

www.EngineeringBooksPdf.com

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

17.4.5 CBF Format Examples

Minimal Working Example

The conic optimization problem (17.6) , has three variables in a quadratic cone - first one is integer - and an affine expression in domain 0 (equality constraint).

$$\begin{array}{ll} \text{minimize} & 5.1\,x_0 \\ \text{subject to} & 6.2\,x_1 + 7.3\,x_2 - 8.4 \in \{0\} \\ & x \in \mathcal{Q}^3,\ x_0 \in \mathbb{Z}. \end{array} \tag{17.6}$$

Its formulation in the Conic Benchmark Format begins with the version of the CBF format used, to safeguard against later revisions.

```
VER
1
```

Next follows the problem structure, consisting of the objective sense, the number and domain of variables, the indices of integer variables, and the number and domain of scalar-valued affine expressions (i.e., the equality constraint).

```
OBJSENSE
MIN

VAR
3 1
Q 3

INT
1
0

CON
1 1
L= 1
```

Finally follows the problem data, consisting of the coefficients of the objective, the coefficients of the constraints, and the constant terms of the constraints. All data is specified on a sparse coordinate form.

```
OBJACOORD
1
0 5.1

ACOORD
2
0 1 6.2
0 2 7.3

BCOORD
1
0 -8.4
```

This concludes the example.

www.EngineeringBooksPdf.com

Mixing Linear, Second-order and Semidefinite Cones

The conic optimization problem (17.7), has a semidefinite cone, a quadratic cone over unordered subindices, and two equality constraints.

$$\begin{array}{lll} \text{minimize} & \left\langle \begin{bmatrix} 2 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 2 \end{bmatrix}, X_1 \right\rangle + x_1 & \\ \text{subject to} & \left\langle \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, X_1 \right\rangle + x_1 & = \quad 1.0\,, \\ & \left\langle \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, X_1 \right\rangle + x_0 + x_2 & = \quad 0.5\,, \\ & x_1 \geq \sqrt{x_0^2 + x_2^2}\,, & \\ & X_1 \succeq \mathbf{0}\,. & \end{array} \tag{17.7}$$

The equality constraints are easily rewritten to the conic form, $(g_0, g_1) \in \{0\}^2$, by moving constants such that the right-hand-side becomes zero. The quadratic cone does not fit under the `VAR` keyword in this variable permutation. Instead, it takes a scalar constraint $(g_2, g_3, g_4) = (x_1, x_0, x_2) \in \mathcal{Q}^3$, with scalar variables constructed as $(x_0, x_1, x_2) \in \mathbb{R}^3$. Its formulation in the CBF format is reported in the following list

```
# File written using this version of the Conic Benchmark Format:
#     | Version 1.
VER
1

# The sense of the objective is:
#     | Minimize.
OBJSENSE
MIN

# One PSD variable of this size:
#     | Three times three.
PSDVAR
1
3

# Three scalar variables in this one conic domain:
#     | Three are free.
VAR
3 1
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
```


www.EngineeringBooksPdf.com

```
0 2 1 1.0
0 2 2 2.0

# One coordinate in a^{obj}_j coefficients:
#     | a^{obj}[1] = 1.0
OBJACOORD
1
1 1.0

# Nine coordinates in F_ij coefficients:
#     | F[0,0][0,0] = 1.0
#     | F[0,0][1,1] = 1.0
#     | and more...
FCOORD
9
0 0 0 0 1.0
0 0 1 1 1.0
0 0 2 2 1.0
1 0 0 0 1.0
1 0 1 0 1.0
1 0 2 0 1.0
1 0 1 1 1.0
1 0 2 1 1.0
1 0 2 2 1.0

# Six coordinates in a_ij coefficients:
#     | a[0,1] = 1.0
#     | a[1,0] = 1.0
#     | and more...
ACOORD
6
0 1 1.0
1 0 1.0
1 2 1.0
2 1 1.0
3 0 1.0
4 2 1.0

# Two coordinates in b_i coefficients:
#     | b[0] = -1.0
#     | b[1] = -0.5
BCOORD
2
0 -1.0
1 -0.5
```

Mixing Semidefinite Variables and Linear Matrix Inequalities

The standard forms in semidefinite optimization are usually based either on semidefinite variables or linear matrix inequalities. In the CBF format, both forms are supported and can even be mixed as shown in.

$$\begin{array}{lll} \text{minimize} & \left\langle \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, X_1 \right\rangle + x_1 + x_2 + 1 & \\ \text{subject to} & \left\langle \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, X_1 \right\rangle - x_1 - x_2 & \geq \quad 0.0\,, \\ & x_1 \begin{bmatrix} 0 & 1 \\ 1 & 3 \end{bmatrix} + x_2 \begin{bmatrix} 3 & 1 \\ 1 & 0 \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \succeq \quad \mathbf{0}\,, \\ & X_1 \succeq \mathbf{0}\,. & \end{array} \tag{17.8}$$

Its formulation in the CBF format is written in what follows

www.EngineeringBooksPdf.com

```
# 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:
#     | 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
```


www.EngineeringBooksPdf.com

```
#     | 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 (17.9), is defined for a sequence of objectives such that hotstarting from one to the next might be advantages.

$$\begin{array}{lll} \text{maximize}_k & g_k^{obj} & \\ \text{subject to} & 50\,x_0 + 31 & \leq \quad 250\,, \\ & 3\,x_0 - 2x_1 & \geq \quad -4\,, \\ & x \in \mathbb{R}^2_+, & \end{array} \tag{17.9}$$

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 17.5.

Listing 17.5: Problem (17.9) 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
```


www.EngineeringBooksPdf.com

```
# 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

# 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
```


www.EngineeringBooksPdf.com

17.5 The XML (OSiL) Format

MOSEK can write data in the standard OSiL xml format. For a definition of the OSiL format please see http://www.optimizationservices.org/.

Only linear constraints (possibly with integer variables) are supported. By default output files with the extension `.xml` are written in the OSiL format.

The parameter *iparam.write_xml_mode* controls if the linear coefficients in the A matrix are written in row or column order.

17.6 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 quadratic, 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:

- The task format *does not* support General Convex problems since these are defined by arbitrary user-defined functions.
- Status of a solution read from a file will *always* be unknown.

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.

17.7 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*

Warning: 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.

www.EngineeringBooksPdf.com

17.7.1 *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**.

17.7.2 *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**.

17.7.3 A *jtask* example

In Listing 17.6 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 17.6: 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",
                "x3",
                "x4"
            ],
            "bk":[
                "lo",
                "ra",
                "lo",
                "lo"
            ],
            "bl":[
                0.0,
                0.0,
                0.0,
                0.0
            ],
            "bu":[
                1e+30,
                1e+1,
                1e+30,
```


www.EngineeringBooksPdf.com

```
                1e+30
            ],
            "type":[
                "cont",
                "cont",
                "cont",
                "cont"
            ]
        },
        "con":{
            "name":[
                "c1",
                "c2",
                "c3"
            ],
            "bk":[
                "fx",
                "lo",
                "up"
            ],
            "bl":[
                3e+1,
                1.5e+1,
                    -1e+30
            ],
            "bu":[
                3e+1,
                1e+30,
                2.5e+1
            ]
        },
        "objective":{
            "sense":"max",
            "name":"obj",
            "c":{
                "subj":[
                    0,
                    1,
                    2,
                    3
                ],
                "val":[
                    3e+0,
                    1e+0,
                    5e+0,
                    1e+0
                ]
            },
            "cfix":0.0
        },
        "A":{
            "subi":[
                0,
                0,
                0,
                1,
                1,
                1,
                1,
                2,
                2
            ],
            "subj":[
```


www.EngineeringBooksPdf.com

```
                0,
                1,
                2,
                0,
                1,
                2,
                3,
                1,
                3
            ],
            "val":[
                3e+0,
                1e+0,
                2e+0,
                2e+0,
                1e+0,
                3e+0,
                1e+0,
                2e+0,
                3e+0
            ]
        }
    },
    "Task/parameters":{
        "iparam":{
            "ANA_SOL_BASIS":"ON",
            "ANA_SOL_PRINT_VIOLATED":"OFF",
            "AUTO_SORT_A_BEFORE_OPT":"OFF",
            "AUTO_UPDATE_SOL_INFO":"OFF",
            "BASIS_SOLVE_USE_PLUS_ONE":"OFF",
            "BI_CLEAN_OPTIMIZER":"OPTIMIZER_FREE",
            "BI_IGNORE_MAX_ITER":"OFF",
            "BI_IGNORE_NUM_ERROR":"OFF",
            "BI_MAX_ITERATIONS":1000000,
            "CACHE_LICENSE":"ON",
            "CHECK_CONVEXITY":"CHECK_CONVEXITY_FULL",
            "COMPRESS_STATFILE":"ON",
            "CONCURRENT_NUM_OPTIMIZERS":2,
            "CONCURRENT_PRIORITY_DUAL_SIMPLEX":2,
            "CONCURRENT_PRIORITY_FREE_SIMPLEX":3,
            "CONCURRENT_PRIORITY_INTPNT":4,
            "CONCURRENT_PRIORITY_PRIMAL_SIMPLEX":1,
            "FEASREPAIR_OPTIMIZE":"FEASREPAIR_OPTIMIZE_NONE",
            "INFEAS_GENERIC_NAMES":"OFF",
            "INFEAS_PREFER_PRIMAL":"ON",
            "INFEAS_REPORT_AUTO":"OFF",
            "INFEAS_REPORT_LEVEL":1,
            "INTPNT_BASIS":"BI_ALWAYS",
            "INTPNT_DIFF_STEP":"ON",
            "INTPNT_FACTOR_DEBUG_LVL":0,
            "INTPNT_FACTOR_METHOD":0,
            "INTPNT_HOTSTART":"INTPNT_HOTSTART_NONE",
            "INTPNT_MAX_ITERATIONS":400,
            "INTPNT_MAX_NUM_COR":-1,
            "INTPNT_MAX_NUM_REFINEMENT_STEPS":-1,
            "INTPNT_OFF_COL_TRH":40,
            "INTPNT_ORDER_METHOD":"ORDER_METHOD_FREE",
            "INTPNT_REGULARIZATION_USE":"ON",
            "INTPNT_SCALING":"SCALING_FREE",
            "INTPNT_SOLVE_FORM":"SOLVE_FREE",
            "INTPNT_STARTING_POINT":"STARTING_POINT_FREE",
            "LIC_TRH_EXPIRY_WRN":7,
            "LICENSE_DEBUG":"OFF",
```


www.EngineeringBooksPdf.com

```
            "LICENSE_PAUSE_TIME":0,
            "LICENSE_SUPPRESS_EXPIRE_WRNS":"OFF",
            "LICENSE_WAIT":"OFF",
            "LOG":10,
            "LOG_ANA_PRO":1,
            "LOG_BI":4,
            "LOG_BI_FREQ":2500,
            "LOG_CHECK_CONVEXITY":0,
            "LOG_CONCURRENT":1,
            "LOG_CUT_SECOND_OPT":1,
            "LOG_EXPAND":0,
            "LOG_FACTOR":1,
            "LOG_FEAS_REPAIR":1,
            "LOG_FILE":1,
            "LOG_HEAD":1,
            "LOG_INFEAS_ANA":1,
            "LOG_INTPNT":4,
            "LOG_MIO":4,
            "LOG_MIO_FREQ":1000,
            "LOG_OPTIMIZER":1,
            "LOG_ORDER":1,
            "LOG_PRESOLVE":1,
            "LOG_RESPONSE":0,
            "LOG_SENSITIVITY":1,
            "LOG_SENSITIVITY_OPT":0,
            "LOG_SIM":4,
            "LOG_SIM_FREQ":1000,
            "LOG_SIM_MINOR":1,
            "LOG_STORAGE":1,
            "MAX_NUM_WARNINGS":10,
            "MIO_BRANCH_DIR":"BRANCH_DIR_FREE",
            "MIO_CONSTRUCT_SOL":"OFF",
            "MIO_CUT_CLIQUE":"ON",
            "MIO_CUT_CMIR":"ON",
            "MIO_CUT_GMI":"ON",
            "MIO_CUT_KNAPSACK_COVER":"OFF",
            "MIO_HEURISTIC_LEVEL":-1,
            "MIO_MAX_NUM_BRANCHES":-1,
            "MIO_MAX_NUM_RELAXS":-1,
            "MIO_MAX_NUM_SOLUTIONS":-1,
            "MIO_MODE":"MIO_MODE_SATISFIED",
            "MIO_MT_USER_CB":"ON",
            "MIO_NODE_OPTIMIZER":"OPTIMIZER_FREE",
            "MIO_NODE_SELECTION":"MIO_NODE_SELECTION_FREE",
            "MIO_PERSPECTIVE_REFORMULATE":"ON",
            "MIO_PROBING_LEVEL":-1,
            "MIO_RINS_MAX_NODES":-1,
            "MIO_ROOT_OPTIMIZER":"OPTIMIZER_FREE",
            "MIO_ROOT_REPEAT_PRESOLVE_LEVEL":-1,
            "MT_SPINCOUNT":0,
            "NUM_THREADS":0,
            "OPF_MAX_TERMS_PER_LINE":5,
            "OPF_WRITE_HEADER":"ON",
            "OPF_WRITE_HINTS":"ON",
            "OPF_WRITE_PARAMETERS":"OFF",
            "OPF_WRITE_PROBLEM":"ON",
            "OPF_WRITE_SOL_BAS":"ON",
            "OPF_WRITE_SOL_ITG":"ON",
            "OPF_WRITE_SOL_ITR":"ON",
            "OPF_WRITE_SOLUTIONS":"OFF",
            "OPTIMIZER":"OPTIMIZER_FREE",
            "PARAM_READ_CASE_NAME":"ON",
            "PARAM_READ_IGN_ERROR":"OFF",
```


www.EngineeringBooksPdf.com

```
            "PRESOLVE_ELIMINATOR_MAX_FILL":-1,
            "PRESOLVE_ELIMINATOR_MAX_NUM_TRIES":-1,
            "PRESOLVE_LEVEL":-1,
            "PRESOLVE_LINDEP_ABS_WORK_TRH":100,
            "PRESOLVE_LINDEP_REL_WORK_TRH":100,
            "PRESOLVE_LINDEP_USE":"ON",
            "PRESOLVE_MAX_NUM_REDUCTIONS":-1,
            "PRESOLVE_USE":"PRESOLVE_MODE_FREE",
            "PRIMAL_REPAIR_OPTIMIZER":"OPTIMIZER_FREE",
            "QO_SEPARABLE_REFORMULATION":"OFF",
            "READ_DATA_COMPRESSED":"COMPRESS_FREE",
            "READ_DATA_FORMAT":"DATA_FORMAT_EXTENSION",
            "READ_DEBUG":"OFF",
            "READ_KEEP_FREE_CON":"OFF",
            "READ_LP_DROP_NEW_VARS_IN_BOU":"OFF",
            "READ_LP_QUOTED_NAMES":"ON",
            "READ_MPS_FORMAT":"MPS_FORMAT_FREE",
            "READ_MPS_WIDTH":1024,
            "READ_TASK_IGNORE_PARAM":"OFF",
            "SENSITIVITY_ALL":"OFF",
            "SENSITIVITY_OPTIMIZER":"OPTIMIZER_FREE_SIMPLEX",
            "SENSITIVITY_TYPE":"SENSITIVITY_TYPE_BASIS",
            "SIM_BASIS_FACTOR_USE":"ON",
            "SIM_DEGEN":"SIM_DEGEN_FREE",
            "SIM_DUAL_CRASH":90,
            "SIM_DUAL_PHASEONE_METHOD":0,
            "SIM_DUAL_RESTRICT_SELECTION":50,
            "SIM_DUAL_SELECTION":"SIM_SELECTION_FREE",
            "SIM_EXPLOIT_DUPVEC":"SIM_EXPLOIT_DUPVEC_OFF",
            "SIM_HOTSTART":"SIM_HOTSTART_FREE",
            "SIM_HOTSTART_LU":"ON",
            "SIM_INTEGER":0,
            "SIM_MAX_ITERATIONS":10000000,
            "SIM_MAX_NUM_SETBACKS":250,
            "SIM_NON_SINGULAR":"ON",
            "SIM_PRIMAL_CRASH":90,
            "SIM_PRIMAL_PHASEONE_METHOD":0,
            "SIM_PRIMAL_RESTRICT_SELECTION":50,
            "SIM_PRIMAL_SELECTION":"SIM_SELECTION_FREE",
            "SIM_REFACTOR_FREQ":0,
            "SIM_REFORMULATION":"SIM_REFORMULATION_OFF",
            "SIM_SAVE_LU":"OFF",
            "SIM_SCALING":"SCALING_FREE",
            "SIM_SCALING_METHOD":"SCALING_METHOD_POW2",
            "SIM_SOLVE_FORM":"SOLVE_FREE",
            "SIM_STABILITY_PRIORITY":50,
            "SIM_SWITCH_OPTIMIZER":"OFF",
            "SOL_FILTER_KEEP_BASIC":"OFF",
            "SOL_FILTER_KEEP_RANGED":"OFF",
            "SOL_READ_NAME_WIDTH":-1,
            "SOL_READ_WIDTH":1024,
            "SOLUTION_CALLBACK":"OFF",
            "TIMING_LEVEL":1,
            "WRITE_BAS_CONSTRAINTS":"ON",
            "WRITE_BAS_HEAD":"ON",
            "WRITE_BAS_VARIABLES":"ON",
            "WRITE_DATA_COMPRESSED":0,
            "WRITE_DATA_FORMAT":"DATA_FORMAT_EXTENSION",
            "WRITE_DATA_PARAM":"OFF",
            "WRITE_FREE_CON":"OFF",
            "WRITE_GENERIC_NAMES":"OFF",
            "WRITE_GENERIC_NAMES_IO":1,
            "WRITE_IGNORE_INCOMPATIBLE_CONIC_ITEMS":"OFF",
```


www.EngineeringBooksPdf.com

```
            "WRITE_IGNORE_INCOMPATIBLE_ITEMS":"OFF",
            "WRITE_IGNORE_INCOMPATIBLE_NL_ITEMS":"OFF",
            "WRITE_IGNORE_INCOMPATIBLE_PSD_ITEMS":"OFF",
            "WRITE_INT_CONSTRAINTS":"ON",
            "WRITE_INT_HEAD":"ON",
            "WRITE_INT_VARIABLES":"ON",
            "WRITE_LP_FULL_OBJ":"ON",
            "WRITE_LP_LINE_WIDTH":80,
            "WRITE_LP_QUOTED_NAMES":"ON",
            "WRITE_LP_STRICT_FORMAT":"OFF",
            "WRITE_LP_TERMS_PER_LINE":10,
            "WRITE_MPS_FORMAT":"MPS_FORMAT_FREE",
            "WRITE_MPS_INT":"ON",
            "WRITE_PRECISION":15,
            "WRITE_SOL_BARVARIABLES":"ON",
            "WRITE_SOL_CONSTRAINTS":"ON",
            "WRITE_SOL_HEAD":"ON",
            "WRITE_SOL_IGNORE_INVALID_NAMES":"OFF",
            "WRITE_SOL_VARIABLES":"ON",
            "WRITE_TASK_INC_SOL":"ON",
            "WRITE_XML_MODE":"WRITE_XML_MODE_ROW"
        },
        "dparam":{
            "ANA_SOL_INFEAS_TOL":1e-6,
            "BASIS_REL_TOL_S":1e-12,
            "BASIS_TOL_S":1e-6,
            "BASIS_TOL_X":1e-6,
            "CHECK_CONVEXITY_REL_TOL":1e-10,
            "DATA_TOL_AIJ":1e-12,
            "DATA_TOL_AIJ_HUGE":1e+20,
            "DATA_TOL_AIJ_LARGE":1e+10,
            "DATA_TOL_BOUND_INF":1e+16,
            "DATA_TOL_BOUND_WRN":1e+8,
            "DATA_TOL_C_HUGE":1e+16,
            "DATA_TOL_CJ_LARGE":1e+8,
            "DATA_TOL_QIJ":1e-16,
            "DATA_TOL_X":1e-8,
            "FEASREPAIR_TOL":1e-10,
            "INTPNT_CO_TOL_DFEAS":1e-8,
            "INTPNT_CO_TOL_INFEAS":1e-10,
            "INTPNT_CO_TOL_MU_RED":1e-8,
            "INTPNT_CO_TOL_NEAR_REL":1e+3,
            "INTPNT_CO_TOL_PFEAS":1e-8,
            "INTPNT_CO_TOL_REL_GAP":1e-7,
            "INTPNT_NL_MERIT_BAL":1e-4,
            "INTPNT_NL_TOL_DFEAS":1e-8,
            "INTPNT_NL_TOL_MU_RED":1e-12,
            "INTPNT_NL_TOL_NEAR_REL":1e+3,
            "INTPNT_NL_TOL_PFEAS":1e-8,
            "INTPNT_NL_TOL_REL_GAP":1e-6,
            "INTPNT_NL_TOL_REL_STEP":9.95e-1,
            "INTPNT_QO_TOL_DFEAS":1e-8,
            "INTPNT_QO_TOL_INFEAS":1e-10,
            "INTPNT_QO_TOL_MU_RED":1e-8,
            "INTPNT_QO_TOL_NEAR_REL":1e+3,
            "INTPNT_QO_TOL_PFEAS":1e-8,
            "INTPNT_QO_TOL_REL_GAP":1e-8,
            "INTPNT_TOL_DFEAS":1e-8,
            "INTPNT_TOL_DSAFE":1e+0,
            "INTPNT_TOL_INFEAS":1e-10,
            "INTPNT_TOL_MU_RED":1e-16,
            "INTPNT_TOL_PATH":1e-8,
            "INTPNT_TOL_PFEAS":1e-8,
```


www.EngineeringBooksPdf.com

```
            "INTPNT_TOL_PSAFE":1e+0,
            "INTPNT_TOL_REL_GAP":1e-8,
            "INTPNT_TOL_REL_STEP":9.999e-1,
            "INTPNT_TOL_STEP_SIZE":1e-6,
            "LOWER_OBJ_CUT":-1e+30,
            "LOWER_OBJ_CUT_FINITE_TRH":-5e+29,
            "MIO_DISABLE_TERM_TIME":-1e+0,
            "MIO_MAX_TIME":-1e+0,
            "MIO_MAX_TIME_APRX_OPT":6e+1,
            "MIO_NEAR_TOL_ABS_GAP":0.0,
            "MIO_NEAR_TOL_REL_GAP":1e-3,
            "MIO_REL_GAP_CONST":1e-10,
            "MIO_TOL_ABS_GAP":0.0,
            "MIO_TOL_ABS_RELAX_INT":1e-5,
            "MIO_TOL_FEAS":1e-6,
            "MIO_TOL_REL_DUAL_BOUND_IMPROVEMENT":0.0,
            "MIO_TOL_REL_GAP":1e-4,
            "MIO_TOL_X":1e-6,
            "OPTIMIZER_MAX_TIME":-1e+0,
            "PRESOLVE_TOL_ABS_LINDEP":1e-6,
            "PRESOLVE_TOL_AIJ":1e-12,
            "PRESOLVE_TOL_REL_LINDEP":1e-10,
            "PRESOLVE_TOL_S":1e-8,
            "PRESOLVE_TOL_X":1e-8,
            "QCQO_REFORMULATE_REL_DROP_TOL":1e-15,
            "SEMIDEFINITE_TOL_APPROX":1e-10,
            "SIM_LU_TOL_REL_PIV":1e-2,
            "SIMPLEX_ABS_TOL_PIV":1e-7,
            "UPPER_OBJ_CUT":1e+30,
            "UPPER_OBJ_CUT_FINITE_TRH":5e+29
        },
        "sparam":{
            "BAS_SOL_FILE_NAME":"",
            "DATA_FILE_NAME":"examples/tools/data/lo1.mps",
            "DEBUG_FILE_NAME":"",
            "INT_SOL_FILE_NAME":"",
            "ITR_SOL_FILE_NAME":"",
            "MIO_DEBUG_STRING":"",
            "PARAM_COMMENT_SIGN":"%%",
            "PARAM_READ_FILE_NAME":"",
            "PARAM_WRITE_FILE_NAME":"",
            "READ_MPS_BOU_NAME":"",
            "READ_MPS_OBJ_NAME":"",
            "READ_MPS_RAN_NAME":"",
            "READ_MPS_RHS_NAME":"",
            "SENSITIVITY_FILE_NAME":"",
            "SENSITIVITY_RES_FILE_NAME":"",
            "SOL_FILTER_XC_LOW":"",
            "SOL_FILTER_XC_UPR":"",
            "SOL_FILTER_XX_LOW":"",
            "SOL_FILTER_XX_UPR":"",
            "STAT_FILE_NAME":"",
            "STAT_KEY":"",
            "STAT_NAME":"",
            "WRITE_LP_GEN_VAR_NAME":"XMSKGEN"
        }
    }
}
```


www.EngineeringBooksPdf.com

17.8 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

$$l_i^c \leq \sum_{j=1}^{n} a_{ij} x_j \leq u_i^c, \tag{17.10}$$

  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 17.4 the possible values of the status keys and their interpretation are shown.

Table 17.4: 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 (17.10).)
  - `UPPER LIMIT`: the quantity u_i^c (see (17.10).)

www.EngineeringBooksPdf.com

  - 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 17.7 we show the solution file for the `lo1.opf` problem.

Listing 17.7: 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    -2.49999999741654e+00
1          c2             SB 5.33333333049188e+01    1.50000000e+01     NONE               2.
 ↪09157603759397e-10     -0.00000000000000e+00
2          c3             UL 2.49999999842049e+01    NONE               2.50000000e+01     -0.
 ↪00000000000000e+00    -3.33333332895110e-01

VARIABLES
INDEX      NAME           AT ACTIVITY                LOWER LIMIT        UPPER LIMIT       ␣
 ↪DUAL LOWER              DUAL UPPER
0          x1             LL 1.67020427073508e-09    0.00000000e+00     NONE               -4.
 ↪49999999528055e+00    -0.00000000000000e+00
1          x2             LL 2.93510446280504e-09    0.00000000e+00     1.00000000e+01     -2.
 ↪16666666494916e+00    6.20863861687316e-10
2          x3             SB 1.49999999899425e+01    0.00000000e+00     NONE               -8.
 ↪79123177454657e-10    -0.00000000000000e+00
3          x4             SB 8.33333332273116e+00    0.00000000e+00     NONE               -1.
 ↪69795978899185e-09    -0.00000000000000e+00
```


www.EngineeringBooksPdf.com

CHAPTER

EIGHTEEN

INTERFACE CHANGES

The section show interface-specific changes to the **MOSEK** Optimizer API for Python in version 8. See the release notes for general changes and new features of the **MOSEK** Optimization Suite.

18.1 Compatibility

- All input functions of the form `putXXXlist` now perform strict dimensional checking. That means all input arrays must have the same size. In previous release they were allowed to differs and **MOSEK** would have used the shortest dimension.
- Compatibility guarantees for this interface has been updated. See the new *state of compatibility*.

18.2 Functions

Added

Changed

Removed

- `Env.putdllpath`
- `Env.putkeepdlls`
- `Task.getdbi`
- `Task.getdcni`
- `Task.getdeqi`
- `Task.getinti`
- `Task.getnumqconknz64`
- `Task.getpbi`
- `Task.getpcni`
- `Task.getpeqi`
- `Task.getqobj64`
- `Task.getsolutioninf`
- `Task.getvarbranchdir`
- `Task.getvarbranchpri`
- `Task.optimizeconcurrent`

www.EngineeringBooksPdf.com

- Task.putvarbranchorder
- Task.readbranchpriorities
- Task.relaxprimal
- Task.writebranchpriorities

18.3 Parameters

Added

- *dparam.data_sym_mat_tol*
- *dparam.data_sym_mat_tol_huge*
- *dparam.data_sym_mat_tol_large*
- *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.semidefinite_tol_approx*
- *iparam.intpnt_multi_thread*
- *iparam.license_trh_expiry_wrn*
- *iparam.log_ana_pro*
- *iparam.mio_cut_clique*
- *iparam.mio_cut_gmi*
- *iparam.mio_cut_implied_bound*
- *iparam.mio_cut_knapsack_cover*
- *iparam.mio_cut_selection_level*
- *iparam.mio_perspective_reformulate*
- *iparam.mio_root_repeat_presolve_level*
- *iparam.mio_vb_detection_level*
- *iparam.presolve_eliminator_max_fill*

Removed

- dparam.feasrepair_tol
- dparam.mio_heuristic_time
- dparam.mio_max_time_aprx_opt
- dparam.mio_rel_add_cut_limited
- dparam.mio_tol_max_cut_frac_rhs
- dparam.mio_tol_min_cut_frac_rhs

www.EngineeringBooksPdf.com

- dparam.mio_tol_rel_relax_int
- dparam.mio_tol_x
- dparam.nonconvex_tol_feas
- dparam.nonconvex_tol_opt
- iparam.alloc_add_qnz
- iparam.concurrent_num_optimizers
- iparam.concurrent_priority_dual_simplex
- iparam.concurrent_priority_free_simplex
- iparam.concurrent_priority_intpnt
- iparam.concurrent_priority_primal_simplex
- iparam.feasrepair_optimize
- iparam.intpnt_factor_debug_lvl
- iparam.intpnt_factor_method
- iparam.lic_trh_expiry_wrn
- iparam.log_concurrent
- iparam.log_nonconvex
- iparam.log_param
- iparam.log_sim_network_freq
- iparam.mio_branch_priorities_use
- iparam.mio_cont_sol
- iparam.mio_cut_cg
- iparam.mio_cut_level_root
- iparam.mio_cut_level_tree
- iparam.mio_feaspump_level
- iparam.mio_hotstart
- iparam.mio_keep_basis
- iparam.mio_local_branch_number
- iparam.mio_optimizer_mode
- iparam.mio_presolve_aggregate
- iparam.mio_presolve_probing
- iparam.mio_presolve_use
- iparam.mio_strong_branch
- iparam.mio_use_multithreaded_optimizer
- iparam.nonconvex_max_iterations
- iparam.presolve_elim_fill
- iparam.presolve_eliminator_use
- iparam.qo_separable_reformulation
- iparam.read_anz
- iparam.read_con

www.EngineeringBooksPdf.com

- iparam.read_cone
- iparam.read_mps_keep_int
- iparam.read_mps_obj_sense
- iparam.read_mps_relax
- iparam.read_qnz
- iparam.read_var
- iparam.warning_level
- iparam.write_ignore_incompatible_conic_items
- iparam.write_ignore_incompatible_nl_items
- iparam.write_ignore_incompatible_psd_items
- sparam.feasrepair_name_prefix
- sparam.feasrepair_name_separator
- sparam.feasrepair_name_wsumviol

18.4 Constants

Added

- *branchdir.far*
- *branchdir.guided*
- *branchdir.near*
- *branchdir.pseudocost*
- *branchdir.root_lp*
- *callbackcode.begin_root_cutgen*
- *callbackcode.begin_to_conic*
- *callbackcode.end_root_cutgen*
- *callbackcode.end_to_conic*
- *callbackcode.im_root_cutgen*
- *callbackcode.solving_remote*
- *dataformat.json_task*
- *dinfitem.mio_clique_separation_time*
- *dinfitem.mio_cmir_separation_time*
- *dinfitem.mio_gmi_separation_time*
- *dinfitem.mio_implied_bound_time*
- *dinfitem.mio_knapsack_cover_separation_time*
- *dinfitem.qcqo_reformulate_max_perturbation*
- *dinfitem.qcqo_reformulate_worst_cholesky_column_scaling*
- *dinfitem.qcqo_reformulate_worst_cholesky_diag_scaling*
- *dinfitem.sol_bas_nrm_barx*

www.EngineeringBooksPdf.com

- *dinfitem.sol_bas_nrm_slc*
- *dinfitem.sol_bas_nrm_slx*
- *dinfitem.sol_bas_nrm_suc*
- *dinfitem.sol_bas_nrm_sux*
- *dinfitem.sol_bas_nrm_xc*
- *dinfitem.sol_bas_nrm_xx*
- *dinfitem.sol_bas_nrm_y*
- *dinfitem.sol_itg_nrm_barx*
- *dinfitem.sol_itg_nrm_xc*
- *dinfitem.sol_itg_nrm_xx*
- *dinfitem.sol_itr_nrm_bars*
- *dinfitem.sol_itr_nrm_barx*
- *dinfitem.sol_itr_nrm_slc*
- *dinfitem.sol_itr_nrm_slx*
- *dinfitem.sol_itr_nrm_snx*
- *dinfitem.sol_itr_nrm_suc*
- *dinfitem.sol_itr_nrm_sux*
- *dinfitem.sol_itr_nrm_xc*
- *dinfitem.sol_itr_nrm_xx*
- *dinfitem.sol_itr_nrm_y*
- *dinfitem.to_conic_time*
- *iinfitem.mio_absgap_satisfied*
- *iinfitem.mio_clique_table_size*
- *iinfitem.mio_near_absgap_satisfied*
- *iinfitem.mio_near_relgap_satisfied*
- *iinfitem.mio_node_depth*
- *iinfitem.mio_num_cmir_cuts*
- *iinfitem.mio_num_implied_bound_cuts*
- *iinfitem.mio_num_knapsack_cover_cuts*
- *iinfitem.mio_num_repeated_presolve*
- *iinfitem.mio_presolved_numbin*
- *iinfitem.mio_presolved_numcon*
- *iinfitem.mio_presolved_numcont*
- *iinfitem.mio_presolved_numint*
- *iinfitem.mio_presolved_numvar*
- *iinfitem.mio_relgap_satisfied*
- *liinfitem.mio_presolved_anz*
- *liinfitem.mio_sim_maxiter_setbacks*
- *mpsformat.cplex*

www.EngineeringBooksPdf.com

- *solsta.dual_illposed_cer*
- *solsta.prim_illposed_cer*

Changed

- *solsta.integer_optimal*
- *solsta.near_integer_optimal*

Removed

- constant.callbackcode.begin_concurrent
- constant.callbackcode.begin_network_dual_simplex
- constant.callbackcode.begin_network_primal_simplex
- constant.callbackcode.begin_network_simplex
- constant.callbackcode.begin_nonconvex
- constant.callbackcode.begin_simplex_network_detect
- constant.callbackcode.end_concurrent
- constant.callbackcode.end_network_dual_simplex
- constant.callbackcode.end_network_primal_simplex
- constant.callbackcode.end_network_simplex
- constant.callbackcode.end_nonconvex
- constant.callbackcode.end_simplex_network_detect
- constant.callbackcode.im_mio_presolve
- constant.callbackcode.im_network_dual_simplex
- constant.callbackcode.im_network_primal_simplex
- constant.callbackcode.im_nonconvex
- constant.callbackcode.noncovex
- constant.callbackcode.update_network_dual_simplex
- constant.callbackcode.update_network_primal_simplex
- constant.callbackcode.update_nonconvex
- constant.dinfitem.concurrent_time
- constant.dinfitem.mio_cg_seperation_time
- constant.dinfitem.mio_cmir_seperation_time
- constant.dinfitem.sim_network_dual_time
- constant.dinfitem.sim_network_primal_time
- constant.dinfitem.sim_network_time
- constant.feature.ptom
- constant.feature.ptox
- constant.iinfitem.concurrent_fastest_optimizer
- constant.iinfitem.mio_num_basis_cuts

www.EngineeringBooksPdf.com

- constant.iinfitem.mio_num_cardgub_cuts
- constant.iinfitem.mio_num_coef_redc_cuts
- constant.iinfitem.mio_num_contra_cuts
- constant.iinfitem.mio_num_disagg_cuts
- constant.iinfitem.mio_num_flow_cover_cuts
- constant.iinfitem.mio_num_gcd_cuts
- constant.iinfitem.mio_num_gub_cover_cuts
- constant.iinfitem.mio_num_knapsur_cover_cuts
- constant.iinfitem.mio_num_lattice_cuts
- constant.iinfitem.mio_num_lift_cuts
- constant.iinfitem.mio_num_obj_cuts
- constant.iinfitem.mio_num_plan_loc_cuts
- constant.iinfitem.sim_network_dual_deg_iter
- constant.iinfitem.sim_network_dual_hotstart
- constant.iinfitem.sim_network_dual_hotstart_lu
- constant.iinfitem.sim_network_dual_inf_iter
- constant.iinfitem.sim_network_dual_iter
- constant.iinfitem.sim_network_primal_deg_iter
- constant.iinfitem.sim_network_primal_hotstart
- constant.iinfitem.sim_network_primal_hotstart_lu
- constant.iinfitem.sim_network_primal_inf_iter
- constant.iinfitem.sim_network_primal_iter
- constant.iinfitem.sol_int_prosta
- constant.iinfitem.sol_int_solsta
- constant.iinfitem.sto_num_a_cache_flushes
- constant.iinfitem.sto_num_a_transposes
- constant.miomode.lazy
- constant.optimizertype.concurrent
- constant.optimizertype.mixed_int_conic
- constant.optimizertype.network_primal_simplex
- constant.optimizertype.nonconvex
- constant.optimizertype.primal_dual_simplex

18.5 Response Codes

Added

- *rescode.err_duplicate_aij (1385)*
- *rescode.err_json_data (1179)*

www.EngineeringBooksPdf.com

- *rescode.err_json_format (1178)*
- *rescode.err_json_missing_data (1180)*
- *rescode.err_json_number_overflow (1177)*
- *rescode.err_json_string (1176)*
- *rescode.err_json_syntax (1175)*
- *rescode.err_lau_invalid_lower_triangular_matrix (7002)*
- *rescode.err_lau_invalid_sparse_symmetric_matrix (7019)*
- *rescode.err_lau_not_positive_definite (7001)*
- *rescode.err_mixed_conic_and_nl (1501)*
- *rescode.err_server_connect (8000)*
- *rescode.err_server_protocol (8001)*
- *rescode.err_server_status (8002)*
- *rescode.err_server_token (8003)*
- *rescode.err_sym_mat_huge (1482)*
- *rescode.err_sym_mat_invalid (1480)*
- *rescode.err_task_write (2562)*
- *rescode.err_toconic_constr_not_conic (7153)*
- *rescode.err_toconic_constr_q_not_psd (7150)*
- *rescode.err_toconic_constraint_fx (7151)*
- *rescode.err_toconic_constraint_ra (7152)*
- *rescode.err_toconic_objective_not_psd (7155)*
- *rescode.wrn_sym_mat_large (960)*

Removed

- rescode.err_ad_invalid_operand
- rescode.err_ad_invalid_operator
- rescode.err_ad_missing_operand
- rescode.err_ad_missing_return
- rescode.err_concurrent_optimizer
- rescode.err_inv_conic_problem
- rescode.err_invalid_branch_direction
- rescode.err_invalid_branch_priority
- rescode.err_invalid_network_problem
- rescode.err_mbt_incompatible
- rescode.err_mbt_invalid
- rescode.err_mixed_problem
- rescode.err_no_dual_info_for_itg_sol
- rescode.err_ord_invalid
- rescode.err_ord_invalid_branch_dir

www.EngineeringBooksPdf.com

- rescode.err_toconic_conversion_fail
- rescode.err_too_many_concurrent_tasks
- rescode.wrn_too_many_threads_concurrent

www.EngineeringBooksPdf.com

www.EngineeringBooksPdf.com

[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.

[AY98] E. D. Andersen and Y. Ye. A computational study of the homogeneous algorithm for large-scale convex optimization. *Computational Optimization and Applications*, 10:243–269, 1998.

[AY99] E. D. Andersen and Y. Ye. On a homogeneous algorithm for the monotone complementarity problem. *Math. Programming*, 84(2):375–399, February 1999.

[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.

[BSS93] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty. *Nonlinear programming: Theory and algorithms*. John Wiley and Sons, New York, 2 edition, 1993.

[Chv83] V. Chvátal. *Linear programming*. W.H. Freeman and Company, 1983.

[CT07] Gerard Cornuejols and Reha Tütüncü. *Optimization methods in finance*. Cambridge University Press, New York, 2007.

[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.

www.EngineeringBooksPdf.com

[MOSEKApS12] MOSEK ApS. *The MOSEK Modeling Cookbook*. MOSEK ApS, Fruebjergvej 3, Boks 16, 2100 Copenhagen O, 2012. Last revised September 2015. URL: http://docs.mosek.com/generic/modeling-a4.pdf.

www.EngineeringBooksPdf.com

Classes

Env, 262
Env._del_, 263
Env.axpy, 263
Env.checkinall, 263
Env.checkinlicense, 263
Env.checkoutlicense, 263
Env.computesparsecholesky, 263
Env.dot, 264
Env.echointro, 264
Env.env, 262
Env.gemm, 264
Env.gemv, 265
Env.getcodedesc, 265
Env.getversion, 266
Env.licensecleanup, 266
Env.linkfiletostream, 266
Env.potrf, 266
Env.putlicensecode, 266
Env.putlicensedebug, 266
Env.putlicensepath, 266
Env.putlicensewait, 266
Env.set_stream, 267
Env.sparsetriangularsolvedense, 267
Env.syeig, 267
Env.syevd, 267
Env.syrk, 268
Task, 268
Task.analyzenames, 276
Task.analyzeproblem, 276
Task.analyzesolution, 276
Task.appendbarvars, 276
Task.appendcone, 276
Task.appendconeseq, 277
Task.appendconesseq, 277
Task.appendcons, 277
Task.appendsparsesymmat, 277
Task.appendvars, 278
Task.asyncgetresult, 278
Task.asyncoptimize, 278
Task.asyncpoll, 278
Task.asyncstop, 279
Task.basiscond, 279
Task.checkconvexity, 279
Task.checkmem, 279
Task.chgbound, 279
Task.chgconbound, 280
Task.chgvarbound, 280
Task.commitchanges, 280
Task.deletesolution, 280
Task.dualsensitivity, 281
Task.getacol, 281
Task.getacolnumnz, 281
Task.getacolslicetrip, 281
Task.getaij, 281
Task.getapiecenumnz, 282
Task.getarow, 282
Task.getarownumnz, 282
Task.getarowslicetrip, 282
Task.getaslice, 282
Task.getaslicenumnz, 283
Task.getbarablocktriplet, 283
Task.getbaraidx, 283
Task.getbaraidxij, 283
Task.getbaraidxinfo, 284
Task.getbarasparsity, 284
Task.getbarcblocktriplet, 284
Task.getbarcidx, 284
Task.getbarcidxinfo, 285
Task.getbarcidxj, 285
Task.getbarcsparsity, 285
Task.getbarsj, 285
Task.getbarvarname, 285
Task.getbarvarnameindex, 285
Task.getbarvarnamelen, 285
Task.getbarxj, 286
Task.getbound, 286
Task.getboundslice, 286
Task.getc, 286
Task.getcfix, 286
Task.getcj, 286
Task.getconbound, 287
Task.getconboundslice, 287
Task.getcone, 287
Task.getconeinfo, 287
Task.getconename, 287
Task.getconenameindex, 287
Task.getconenamelen, 288
Task.getconname, 288
Task.getconnameindex, 288
Task.getconnamelen, 288
Task.getcslice, 288
Task.getdimbarvarj, 288

www.EngineeringBooksPdf.com

Task.getdouinf, 289
Task.getdouparam, 289
Task.getdualobj, 289
Task.getdualsolutionnorms, 289
Task.getdviolbarvar, 289
Task.getdviolcon, 289
Task.getdviolcones, 290
Task.getdviolvar, 290
Task.getinfeasiblesubproblem, 290
Task.getintinf, 291
Task.getintparam, 291
Task.getlenbarvarj, 291
Task.getlintinf, 291
Task.getmaxnumanz, 291
Task.getmaxnumbarvar, 291
Task.getmaxnumcon, 291
Task.getmaxnumcone, 292
Task.getmaxnumqnz, 292
Task.getmaxnumvar, 292
Task.getmemusage, 292
Task.getnumanz, 292
Task.getnumanz64, 292
Task.getnumbarablocktriplets, 292
Task.getnumbaranz, 292
Task.getnumbarcblocktriplets, 292
Task.getnumbarcnz, 292
Task.getnumbarvar, 293
Task.getnumcon, 293
Task.getnumcone, 293
Task.getnumconemem, 293
Task.getnumintvar, 293
Task.getnumparam, 293
Task.getnumqconknz, 293
Task.getnumqobjnz, 293
Task.getnumsymmat, 293
Task.getnumvar, 294
Task.getobjname, 294
Task.getobjnamelen, 294
Task.getobjsense, 294
Task.getprimalobj, 294
Task.getprimalsolutionnorms, 294
Task.getprobtype, 294
Task.getprosta, 294
Task.getpviolbarvar, 294
Task.getpviolcon, 295
Task.getpviolcones, 295
Task.getpviolvar, 295
Task.getqconk, 295
Task.getqobj, 296
Task.getqobjij, 296
Task.getreducedcosts, 296
Task.getskc, 296
Task.getskcslice, 296
Task.getskx, 297
Task.getskxslice, 297
Task.getslc, 297
Task.getslcslice, 297
Task.getslx, 297
Task.getslxslice, 297
Task.getsnx, 297
Task.getsnxslice, 297
Task.getsolsta, 298
Task.getsolution, 298
Task.getsolutioni, 299
Task.getsolutioninfo, 299
Task.getsolutionslice, 300
Task.getsparsesymmat, 301
Task.getstrparam, 301
Task.getstrparamlen, 301
Task.getsuc, 301
Task.getsucslice, 301
Task.getsux, 302
Task.getsuxslice, 302
Task.getsymmatinfo, 302
Task.gettaskname, 302
Task.gettasknamelen, 302
Task.getvarbound, 302
Task.getvarboundslice, 302
Task.getvarname, 303
Task.getvarnameindex, 303
Task.getvarnamelen, 303
Task.getvartype, 303
Task.getvartypelist, 303
Task.getxc, 303
Task.getxcslice, 303
Task.getxx, 304
Task.getxxslice, 304
Task.gety, 304
Task.getyslice, 304
Task.initbasissolve, 304
Task.inputdata, 304
Task.isdouparname, 305
Task.isintparname, 305
Task.isstrparname, 305
Task.linkfiletostream, 305
Task.onesolutionsummary, 305
Task.optimize, 306
Task.optimizermt, 306
Task.optimizersummary, 306
Task.primalrepair, 306
Task.primalsensitivity, 307
Task.printdata, 308
Task.putacol, 308
Task.putacollist, 309
Task.putacolslice, 309
Task.putaij, 309
Task.putaijlist, 309
Task.putarow, 310
Task.putarowlist, 310
Task.putarowslice, 310
Task.putbarablocktriplet, 310
Task.putbaraij, 311
Task.putbarcblocktriplet, 311
Task.putbarcj, 311
Task.putbarsj, 311
Task.putbarvarname, 312

www.EngineeringBooksPdf.com

Task.putbarxj, 312
Task.putbound, 312
Task.putboundlist, 312
Task.putboundslice, 312
Task.putcfix, 313
Task.putcj, 313
Task.putclist, 313
Task.putconbound, 313
Task.putconboundlist, 313
Task.putconboundslice, 313
Task.putcone, 314
Task.putconename, 314
Task.putconname, 314
Task.putcslice, 314
Task.putdouparam, 314
Task.putintparam, 314
Task.putmaxnumanz, 314
Task.putmaxnumbarvar, 315
Task.putmaxnumcon, 315
Task.putmaxnumcone, 315
Task.putmaxnumqnz, 315
Task.putmaxnumvar, 315
Task.putnadouparam, 316
Task.putnaintparam, 316
Task.putnastrparam, 316
Task.putobjname, 316
Task.putobjsense, 316
Task.putparam, 316
Task.putqcon, 316
Task.putqconk, 317
Task.putqobj, 317
Task.putqobjij, 318
Task.putskc, 318
Task.putskcslice, 318
Task.putskx, 318
Task.putskxslice, 318
Task.putslc, 319
Task.putslcslice, 319
Task.putslx, 319
Task.putslxslice, 319
Task.putsnx, 319
Task.putsnxslice, 319
Task.putsolution, 319
Task.putsolutioni, 320
Task.putsolutionyi, 320
Task.putstrparam, 320
Task.putsuc, 320
Task.putsucslice, 320
Task.putsux, 321
Task.putsuxslice, 321
Task.puttaskname, 321
Task.putvarbound, 321
Task.putvarboundlist, 321
Task.putvarboundslice, 321
Task.putvarname, 321
Task.putvartype, 322
Task.putvartypelist, 322
Task.putxc, 322
Task.putxcslice, 322
Task.putxx, 322
Task.putxxslice, 322
Task.puty, 322
Task.putyslice, 323
Task.readdata, 323
Task.readdataformat, 323
Task.readparamfile, 323
Task.readsolution, 323
Task.readsummary, 323
Task.readtask, 323
Task.removebarvars, 323
Task.removecones, 323
Task.removecons, 324
Task.removevars, 324
Task.resizetask, 324
Task.sensitivityreport, 324
Task.set_progress, 324
Task.set_stream, 324
Task.setdefaults, 324
Task.solutiondef, 325
Task.solutionsummary, 325
Task.solvewithbasis, 325
Task.strtoconetype, 326
Task.strtosk, 326
Task.task, 275
Task.toconic, 326
Task.updatesolutioninfo, 326
Task.writedata, 326
Task.writejsonsol, 327
Task.writeparamfile, 327
Task.writesc, 326
Task.writesolution, 327
Task.writetask, 327
Task.writetasksolverresult_file, 327

Enumerations

accmode, 235
accmode.con, 235
accmode.var, 235
basindtype, 235
basindtype.always, 235
basindtype.if_feasible, 235
basindtype.never, 235
basindtype.no_error, 235
basindtype.reserved, 235
boundkey, 235
boundkey.fr, 235
boundkey.fx, 235
boundkey.lo, 235
boundkey.ra, 235
boundkey.up, 235
branchdir, 253
branchdir.down, 253
branchdir.far, 253
branchdir.free, 253
branchdir.guided, 253
branchdir.near, 253

www.EngineeringBooksPdf.com

branchdir.pseudocost, 253
branchdir.root_lp, 253
branchdir.up, 253
callbackcode, 237
callbackcode.begin_bi, 238
callbackcode.begin_conic, 238
callbackcode.begin_dual_bi, 238
callbackcode.begin_dual_sensitivity, 241
callbackcode.begin_dual_setup_bi, 241
callbackcode.begin_dual_simplex, 240
callbackcode.begin_dual_simplex_bi, 239
callbackcode.begin_full_convexity_check, 241
callbackcode.begin_infeas_ana, 240
callbackcode.begin_intpnt, 237
callbackcode.begin_license_wait, 241
callbackcode.begin_mio, 239
callbackcode.begin_optimizer, 237
callbackcode.begin_presolve, 237
callbackcode.begin_primal_bi, 238
callbackcode.begin_primal_dual_simplex, 240
callbackcode.begin_primal_dual_simplex_bi, 239
callbackcode.begin_primal_repair, 241
callbackcode.begin_primal_sensitivity, 241
callbackcode.begin_primal_setup_bi, 240
callbackcode.begin_primal_simplex, 240
callbackcode.begin_primal_simplex_bi, 239
callbackcode.begin_qcqo_reformulate, 241
callbackcode.begin_read, 241
callbackcode.begin_root_cutgen, 237
callbackcode.begin_simplex, 240
callbackcode.begin_simplex_bi, 239
callbackcode.begin_to_conic, 241
callbackcode.begin_write, 242
callbackcode.conic, 238
callbackcode.dual_simplex, 238
callbackcode.end_bi, 238
callbackcode.end_conic, 238
callbackcode.end_dual_bi, 238
callbackcode.end_dual_sensitivity, 241
callbackcode.end_dual_setup_bi, 241
callbackcode.end_dual_simplex, 240
callbackcode.end_dual_simplex_bi, 239
callbackcode.end_full_convexity_check, 241
callbackcode.end_infeas_ana, 240
callbackcode.end_intpnt, 238
callbackcode.end_license_wait, 241
callbackcode.end_mio, 239
callbackcode.end_optimizer, 237
callbackcode.end_presolve, 237
callbackcode.end_primal_bi, 238
callbackcode.end_primal_dual_simplex, 240
callbackcode.end_primal_dual_simplex_bi, 239
callbackcode.end_primal_repair, 241
callbackcode.end_primal_sensitivity, 241
callbackcode.end_primal_setup_bi, 241
callbackcode.end_primal_simplex, 240
callbackcode.end_primal_simplex_bi, 239
callbackcode.end_qcqo_reformulate, 241
callbackcode.end_read, 241
callbackcode.end_root_cutgen, 237
callbackcode.end_simplex, 240
callbackcode.end_simplex_bi, 239
callbackcode.end_to_conic, 241
callbackcode.end_write, 242
callbackcode.im_bi, 238
callbackcode.im_conic, 238
callbackcode.im_dual_bi, 238
callbackcode.im_dual_sensivity, 240
callbackcode.im_dual_simplex, 240
callbackcode.im_full_convexity_check, 241
callbackcode.im_intpnt, 237
callbackcode.im_license_wait, 241
callbackcode.im_lu, 242
callbackcode.im_mio, 239
callbackcode.im_mio_dual_simplex, 240
callbackcode.im_mio_intpnt, 240
callbackcode.im_mio_primal_simplex, 240
callbackcode.im_order, 242
callbackcode.im_presolve, 237
callbackcode.im_primal_bi, 238
callbackcode.im_primal_dual_simplex, 240
callbackcode.im_primal_sensivity, 240
callbackcode.im_primal_simplex, 240
callbackcode.im_qo_reformulate, 241
callbackcode.im_read, 241
callbackcode.im_root_cutgen, 237
callbackcode.im_simplex, 242
callbackcode.im_simplex_bi, 239
callbackcode.intpnt, 237
callbackcode.new_int_mio, 239
callbackcode.primal_simplex, 238
callbackcode.read_opf, 242
callbackcode.read_opf_section, 242
callbackcode.solving_remote, 242
callbackcode.update_dual_bi, 238
callbackcode.update_dual_simplex, 240
callbackcode.update_dual_simplex_bi, 239
callbackcode.update_presolve, 237
callbackcode.update_primal_bi, 238
callbackcode.update_primal_dual_simplex, 240
callbackcode.update_primal_dual_simplex_bi, 239
callbackcode.update_primal_simplex, 240
callbackcode.update_primal_simplex_bi, 239
callbackcode.write_opf, 242
checkconvexitytype, 242
checkconvexitytype.full, 242
checkconvexitytype.none, 242
checkconvexitytype.simple, 242
compresstype, 242
compresstype.free, 242
compresstype.gzip, 242

www.EngineeringBooksPdf.com

compresstype.none, 242
conetype, 242
conetype.quad, 242
conetype.rquad, 242
dataformat, 243
dataformat.cb, 243
dataformat.extension, 243
dataformat.free_mps, 243
dataformat.json_task, 243
dataformat.lp, 243
dataformat.mps, 243
dataformat.op, 243
dataformat.task, 243
dataformat.xml, 243
dinfitem, 243
dinfitem.bi_clean_dual_time, 243
dinfitem.bi_clean_primal_dual_time, 243
dinfitem.bi_clean_primal_time, 243
dinfitem.bi_clean_time, 243
dinfitem.bi_dual_time, 243
dinfitem.bi_primal_time, 243
dinfitem.bi_time, 243
dinfitem.intpnt_dual_feas, 244
dinfitem.intpnt_dual_obj, 244
dinfitem.intpnt_factor_num_flops, 247
dinfitem.intpnt_opt_status, 244
dinfitem.intpnt_order_time, 244
dinfitem.intpnt_primal_feas, 244
dinfitem.intpnt_primal_obj, 244
dinfitem.intpnt_time, 244
dinfitem.mio_clique_separation_time, 245
dinfitem.mio_cmir_separation_time, 245
dinfitem.mio_construct_solution_obj, 244
dinfitem.mio_dual_bound_after_presolve, 245
dinfitem.mio_gmi_separation_time, 245
dinfitem.mio_heuristic_time, 244
dinfitem.mio_implied_bound_time, 245
dinfitem.mio_knapsack_cover_separation_time, 245
dinfitem.mio_obj_abs_gap, 245
dinfitem.mio_obj_bound, 245
dinfitem.mio_obj_int, 245
dinfitem.mio_obj_rel_gap, 245
dinfitem.mio_optimizer_time, 244
dinfitem.mio_probing_time, 245
dinfitem.mio_root_cutgen_time, 245
dinfitem.mio_root_optimizer_time, 244
dinfitem.mio_root_presolve_time, 244
dinfitem.mio_time, 244
dinfitem.mio_user_obj_cut, 245
dinfitem.optimizer_time, 245
dinfitem.presolve_eli_time, 245
dinfitem.presolve_lindep_time, 245
dinfitem.presolve_time, 245
dinfitem.primal_repair_penalty_obj, 248
dinfitem.qcqo_reformulate_max_perturbation, 248
dinfitem.qcqo_reformulate_time, 247
dinfitem.qcqo_reformulate_worst_cholesky_column_scaling, 248
dinfitem.qcqo_reformulate_worst_cholesky_diag_scaling, 248
dinfitem.rd_time, 246
dinfitem.sim_dual_time, 244
dinfitem.sim_feas, 244
dinfitem.sim_obj, 244
dinfitem.sim_primal_dual_time, 244
dinfitem.sim_primal_time, 244
dinfitem.sim_time, 244
dinfitem.sol_bas_dual_obj, 247
dinfitem.sol_bas_dviolcon, 247
dinfitem.sol_bas_dviolvar, 247
dinfitem.sol_bas_nrm_barx, 247
dinfitem.sol_bas_nrm_slc, 247
dinfitem.sol_bas_nrm_slx, 247
dinfitem.sol_bas_nrm_suc, 247
dinfitem.sol_bas_nrm_sux, 247
dinfitem.sol_bas_nrm_xc, 247
dinfitem.sol_bas_nrm_xx, 247
dinfitem.sol_bas_nrm_y, 247
dinfitem.sol_bas_primal_obj, 246
dinfitem.sol_bas_pviolcon, 246
dinfitem.sol_bas_pviolvar, 247
dinfitem.sol_itg_nrm_barx, 247
dinfitem.sol_itg_nrm_xc, 247
dinfitem.sol_itg_nrm_xx, 247
dinfitem.sol_itg_primal_obj, 247
dinfitem.sol_itg_pviolbarvar, 247
dinfitem.sol_itg_pviolcon, 247
dinfitem.sol_itg_pviolcones, 247
dinfitem.sol_itg_pviolitg, 247
dinfitem.sol_itg_pviolvar, 247
dinfitem.sol_itr_dual_obj, 246
dinfitem.sol_itr_dviolbarvar, 246
dinfitem.sol_itr_dviolcon, 246
dinfitem.sol_itr_dviolcones, 246
dinfitem.sol_itr_dviolvar, 246
dinfitem.sol_itr_nrm_bars, 246
dinfitem.sol_itr_nrm_barx, 246
dinfitem.sol_itr_nrm_slc, 246
dinfitem.sol_itr_nrm_slx, 246
dinfitem.sol_itr_nrm_snx, 246
dinfitem.sol_itr_nrm_suc, 246
dinfitem.sol_itr_nrm_sux, 246
dinfitem.sol_itr_nrm_xc, 246
dinfitem.sol_itr_nrm_xx, 246
dinfitem.sol_itr_nrm_y, 246
dinfitem.sol_itr_primal_obj, 246
dinfitem.sol_itr_pviolbarvar, 246
dinfitem.sol_itr_pviolcon, 246
dinfitem.sol_itr_pviolcones, 246
dinfitem.sol_itr_pviolvar, 246
dinfitem.to_conic_time, 244
feature, 248
feature.pton, 248
feature.pts, 248

www.EngineeringBooksPdf.com

iinfitem, 249
iinfitem.ana_pro_num_con, 249
iinfitem.ana_pro_num_con_eq, 249
iinfitem.ana_pro_num_con_fr, 249
iinfitem.ana_pro_num_con_lo, 249
iinfitem.ana_pro_num_con_ra, 249
iinfitem.ana_pro_num_con_up, 249
iinfitem.ana_pro_num_var, 249
iinfitem.ana_pro_num_var_bin, 250
iinfitem.ana_pro_num_var_cont, 249
iinfitem.ana_pro_num_var_eq, 249
iinfitem.ana_pro_num_var_fr, 249
iinfitem.ana_pro_num_var_int, 250
iinfitem.ana_pro_num_var_lo, 249
iinfitem.ana_pro_num_var_ra, 249
iinfitem.ana_pro_num_var_up, 249
iinfitem.intpnt_factor_dim_dense, 250
iinfitem.intpnt_iter, 250
iinfitem.intpnt_num_threads, 252
iinfitem.intpnt_solve_dual, 250
iinfitem.mio_absgap_satisfied, 251
iinfitem.mio_clique_table_size, 250
iinfitem.mio_construct_num_roundings, 250
iinfitem.mio_construct_solution, 250
iinfitem.mio_initial_solution, 251
iinfitem.mio_near_absgap_satisfied, 251
iinfitem.mio_near_relgap_satisfied, 251
iinfitem.mio_node_depth, 250
iinfitem.mio_num_active_nodes, 251
iinfitem.mio_num_branch, 251
iinfitem.mio_num_clique_cuts, 251
iinfitem.mio_num_cmir_cuts, 251
iinfitem.mio_num_gomory_cuts, 251
iinfitem.mio_num_implied_bound_cuts, 251
iinfitem.mio_num_int_solutions, 250
iinfitem.mio_num_knapsack_cover_cuts, 251
iinfitem.mio_num_relax, 251
iinfitem.mio_num_repeated_presolve, 251
iinfitem.mio_numcon, 250
iinfitem.mio_numint, 250
iinfitem.mio_numvar, 250
iinfitem.mio_obj_bound_defined, 250
iinfitem.mio_presolved_numbin, 250
iinfitem.mio_presolved_numcon, 250
iinfitem.mio_presolved_numcont, 250
iinfitem.mio_presolved_numint, 250
iinfitem.mio_presolved_numvar, 250
iinfitem.mio_relgap_satisfied, 251
iinfitem.mio_total_num_cuts, 251
iinfitem.mio_user_obj_cut, 251
iinfitem.opt_numcon, 253
iinfitem.opt_numvar, 253
iinfitem.optimize_response, 250
iinfitem.rd_numbarvar, 251
iinfitem.rd_numcon, 251
iinfitem.rd_numcone, 253
iinfitem.rd_numintvar, 251
iinfitem.rd_numq, 251
iinfitem.rd_numvar, 251
iinfitem.rd_protype, 251
iinfitem.sim_dual_deg_iter, 251
iinfitem.sim_dual_hotstart, 252
iinfitem.sim_dual_hotstart_lu, 252
iinfitem.sim_dual_inf_iter, 251
iinfitem.sim_dual_iter, 252
iinfitem.sim_numcon, 252
iinfitem.sim_numvar, 253
iinfitem.sim_primal_deg_iter, 252
iinfitem.sim_primal_dual_deg_iter, 252
iinfitem.sim_primal_dual_hotstart, 252
iinfitem.sim_primal_dual_hotstart_lu, 252
iinfitem.sim_primal_dual_inf_iter, 252
iinfitem.sim_primal_dual_iter, 252
iinfitem.sim_primal_hotstart, 252
iinfitem.sim_primal_hotstart_lu, 252
iinfitem.sim_primal_inf_iter, 252
iinfitem.sim_primal_iter, 252
iinfitem.sim_solve_dual, 253
iinfitem.sol_bas_prosta, 252
iinfitem.sol_bas_solsta, 252
iinfitem.sol_itg_prosta, 252
iinfitem.sol_itg_solsta, 252
iinfitem.sol_itr_prosta, 252
iinfitem.sol_itr_solsta, 252
iinfitem.sto_num_a_realloc, 253
inftype, 253
inftype.dou_type, 253
inftype.int_type, 253
inftype.lint_type, 253
intpnthotstart, 237
intpnthotstart.dual, 237
intpnthotstart.none, 237
intpnthotstart.primal, 237
intpnthotstart.primal_dual, 237
iomode, 253
iomode.read, 253
iomode.readwrite, 253
iomode.write, 253
language, 235
language.dan, 235
language.eng, 235
liinfitem, 248
liinfitem.bi_clean_dual_deg_iter, 248
liinfitem.bi_clean_dual_iter, 248
liinfitem.bi_clean_primal_deg_iter, 248
liinfitem.bi_clean_primal_dual_deg_iter, 248
liinfitem.bi_clean_primal_dual_iter, 248
liinfitem.bi_clean_primal_dual_sub_iter, 248
liinfitem.bi_clean_primal_iter, 248
liinfitem.bi_dual_iter, 248
liinfitem.bi_primal_iter, 248
liinfitem.intpnt_factor_num_nz, 248
liinfitem.mio_intpnt_iter, 248
liinfitem.mio_presolved_anz, 248

www.EngineeringBooksPdf.com

liinfitem.mio_sim_maxiter_setbacks, 249
liinfitem.mio_simplex_iter, 248
liinfitem.rd_numanz, 248
liinfitem.rd_numqnz, 248
mark, 235
mark.lo, 236
mark.up, 236
miocontsoltype, 254
miocontsoltype.itg, 254
miocontsoltype.itg_rel, 254
miocontsoltype.none, 254
miocontsoltype.root, 254
miomode, 254
miomode.ignored, 254
miomode.satisfied, 254
mionodeseltype, 254
mionodeseltype.best, 254
mionodeseltype.first, 254
mionodeseltype.free, 254
mionodeseltype.hybrid, 254
mionodeseltype.pseudo, 254
mionodeseltype.worst, 254
mpsformat, 254
mpsformat.cplex, 254
mpsformat.free, 254
mpsformat.relaxed, 254
mpsformat.strict, 254
msgkey, 255
msgkey.mps_selected, 255
msgkey.reading_file, 255
msgkey.writing_file, 255
nametype, 242
nametype.gen, 242
nametype.lp, 243
nametype.mps, 243
objsense, 255
objsense.maximize, 255
objsense.minimize, 255
onoffkey, 255
onoffkey.off, 255
onoffkey.on, 255
optimizertype, 255
optimizertype.conic, 255
optimizertype.dual_simplex, 255
optimizertype.free, 255
optimizertype.free_simplex, 255
optimizertype.intpnt, 255
optimizertype.mixed_int, 255
optimizertype.primal_simplex, 255
orderingtype, 255
orderingtype.appminloc, 255
orderingtype.experimental, 255
orderingtype.force_graphpar, 255
orderingtype.free, 255
orderingtype.none, 256
orderingtype.try_graphpar, 255
parametertype, 256
parametertype.dou_type, 256
parametertype.int_type, 256
parametertype.invalid_type, 256
parametertype.str_type, 256
presolvemode, 256
presolvemode.free, 256
presolvemode.off, 256
presolvemode.on, 256
problemitem, 256
problemitem.con, 256
problemitem.cone, 256
problemitem.var, 256
problemtype, 256
problemtype.conic, 256
problemtype.geco, 256
problemtype.lo, 256
problemtype.mixed, 256
problemtype.qcqo, 256
problemtype.qo, 256
prosta, 256
prosta.dual_feas, 257
prosta.dual_infeas, 257
prosta.ill_posed, 257
prosta.near_dual_feas, 257
prosta.near_prim_and_dual_feas, 257
prosta.near_prim_feas, 257
prosta.prim_and_dual_feas, 257
prosta.prim_and_dual_infeas, 257
prosta.prim_feas, 257
prosta.prim_infeas, 257
prosta.prim_infeas_or_unbounded, 257
prosta.unknown, 257
rescodetype, 257
rescodetype.err, 257
rescodetype.ok, 257
rescodetype.trm, 257
rescodetype.unk, 257
rescodetype.wrn, 257
scalingmethod, 258
scalingmethod.free, 258
scalingmethod.pow2, 258
scalingtype, 257
scalingtype.aggressive, 258
scalingtype.free, 258
scalingtype.moderate, 258
scalingtype.none, 258
sensitivitytype, 258
sensitivitytype.basis, 258
sensitivitytype.optimal_partition, 258
simdegen, 236
simdegen.aggressive, 236
simdegen.free, 236
simdegen.minimum, 236
simdegen.moderate, 236
simdegen.none, 236
simdupvec, 236
simdupvec.free, 236
simdupvec.off, 236
simdupvec.on, 236

www.EngineeringBooksPdf.com

simhotstart, 237
simhotstart.free, 237
simhotstart.none, 237
simhotstart.status_keys, 237
simreform, 236
simreform.aggressive, 236
simreform.free, 236
simreform.off, 236
simreform.on, 236
simseltype, 258
simseltype.ase, 258
simseltype.devex, 258
simseltype.free, 258
simseltype.full, 258
simseltype.partial, 258
simseltype.se, 258
solitem, 258
solitem.slc, 258
solitem.slx, 259
solitem.snx, 259
solitem.suc, 259
solitem.sux, 259
solitem.xc, 258
solitem.xx, 258
solitem.y, 258
solsta, 259
solsta.dual_feas, 259
solsta.dual_illposed_cer, 259
solsta.dual_infeas_cer, 259
solsta.integer_optimal, 259
solsta.near_dual_feas, 259
solsta.near_dual_infeas_cer, 259
solsta.near_integer_optimal, 259
solsta.near_optimal, 259
solsta.near_prim_and_dual_feas, 259
solsta.near_prim_feas, 259
solsta.near_prim_infeas_cer, 259
solsta.optimal, 259
solsta.prim_and_dual_feas, 259
solsta.prim_feas, 259
solsta.prim_illposed_cer, 259
solsta.prim_infeas_cer, 259
solsta.unknown, 259
soltype, 259
soltype.bas, 260
soltype.itg, 260
soltype.itr, 260
solveform, 260
solveform.dual, 260
solveform.free, 260
solveform.primal, 260
stakey, 260
stakey.bas, 260
stakey.fix, 260
stakey.inf, 260
stakey.low, 260
stakey.supbas, 260
stakey.unk, 260
stakey.upr, 260
startpointtype, 260
startpointtype.constant, 260
startpointtype.free, 260
startpointtype.guess, 260
startpointtype.satisfy_bounds, 260
streamtype, 260
streamtype.err, 261
streamtype.log, 260
streamtype.msg, 261
streamtype.wrn, 261
symmattype, 243
symmattype.sparse, 243
transpose, 236
transpose.no, 236
transpose.yes, 236
uplo, 236
uplo.lo, 236
uplo.up, 236
value, 261
value.license_buffer_length, 261
value.max_str_len, 261
variabletype, 261
variabletype.type_cont, 261
variabletype.type_int, 261
xmlwriteroutputtype, 257
xmlwriteroutputtype.col, 257
xmlwriteroutputtype.row, 257

Exceptions

Error, 327
Exception, 327
Warning, 327

Parameters

Double params, 166
dparam.ana_sol_infeas_tol, 166
dparam.basis_rel_tol_s, 166
dparam.basis_tol_s, 166
dparam.basis_tol_x, 167
dparam.check_convexity_rel_tol, 167
dparam.data_sym_mat_tol, 167
dparam.data_sym_mat_tol_huge, 167
dparam.data_sym_mat_tol_large, 167
dparam.data_tol_aij, 167
dparam.data_tol_aij_huge, 167
dparam.data_tol_aij_large, 168
dparam.data_tol_bound_inf, 168
dparam.data_tol_bound_wrn, 168
dparam.data_tol_c_huge, 168
dparam.data_tol_cj_large, 168
dparam.data_tol_qij, 168
dparam.data_tol_x, 168
dparam.intpnt_co_tol_dfeas, 169
dparam.intpnt_co_tol_infeas, 169
dparam.intpnt_co_tol_mu_red, 169
dparam.intpnt_co_tol_near_rel, 169
dparam.intpnt_co_tol_pfeas, 169

www.EngineeringBooksPdf.com

dparam.intpnt_co_tol_rel_gap, 169
dparam.intpnt_nl_merit_bal, 169
dparam.intpnt_nl_tol_dfeas, 169
dparam.intpnt_nl_tol_mu_red, 170
dparam.intpnt_nl_tol_near_rel, 170
dparam.intpnt_nl_tol_pfeas, 170
dparam.intpnt_nl_tol_rel_gap, 170
dparam.intpnt_nl_tol_rel_step, 170
dparam.intpnt_qo_tol_dfeas, 170
dparam.intpnt_qo_tol_infeas, 170
dparam.intpnt_qo_tol_mu_red, 170
dparam.intpnt_qo_tol_near_rel, 171
dparam.intpnt_qo_tol_pfeas, 171
dparam.intpnt_qo_tol_rel_gap, 171
dparam.intpnt_tol_dfeas, 171
dparam.intpnt_tol_dsafe, 171
dparam.intpnt_tol_infeas, 171
dparam.intpnt_tol_mu_red, 171
dparam.intpnt_tol_path, 172
dparam.intpnt_tol_pfeas, 172
dparam.intpnt_tol_psafe, 172
dparam.intpnt_tol_rel_gap, 172
dparam.intpnt_tol_rel_step, 172
dparam.intpnt_tol_step_size, 172
dparam.lower_obj_cut, 172
dparam.lower_obj_cut_finite_trh, 173
dparam.mio_disable_term_time, 173
dparam.mio_max_time, 173
dparam.mio_near_tol_abs_gap, 173
dparam.mio_near_tol_rel_gap, 173
dparam.mio_rel_gap_const, 173
dparam.mio_tol_abs_gap, 174
dparam.mio_tol_abs_relax_int, 174
dparam.mio_tol_feas, 174
dparam.mio_tol_rel_dual_bound_improvement, 174
dparam.mio_tol_rel_gap, 174
dparam.optimizer_max_time, 174
dparam.presolve_tol_abs_lindep, 174
dparam.presolve_tol_aij, 175
dparam.presolve_tol_rel_lindep, 175
dparam.presolve_tol_s, 175
dparam.presolve_tol_x, 175
dparam.qcqo_reformulate_rel_drop_tol, 175
dparam.semidefinite_tol_approx, 175
dparam.sim_lu_tol_rel_piv, 175
dparam.simplex_abs_tol_piv, 175
dparam.upper_obj_cut, 176
dparam.upper_obj_cut_finite_trh, 176
Integer params, 176
iparam.ana_sol_basis, 176
iparam.ana_sol_print_violated, 176
iparam.auto_sort_a_before_opt, 176
iparam.auto_update_sol_info, 176
iparam.basis_solve_use_plus_one, 176
iparam.bi_clean_optimizer, 177
iparam.bi_ignore_max_iter, 177
iparam.bi_ignore_num_error, 177
iparam.bi_max_iterations, 177
iparam.cache_license, 177
iparam.check_convexity, 177
iparam.compress_statfile, 178
iparam.infeas_generic_names, 178
iparam.infeas_prefer_primal, 178
iparam.infeas_report_auto, 178
iparam.infeas_report_level, 178
iparam.intpnt_basis, 178
iparam.intpnt_diff_step, 178
iparam.intpnt_hotstart, 178
iparam.intpnt_max_iterations, 178
iparam.intpnt_max_num_cor, 179
iparam.intpnt_max_num_refinement_steps, 179
iparam.intpnt_multi_thread, 179
iparam.intpnt_off_col_trh, 179
iparam.intpnt_order_method, 179
iparam.intpnt_regularization_use, 179
iparam.intpnt_scaling, 179
iparam.intpnt_solve_form, 180
iparam.intpnt_starting_point, 180
iparam.license_debug, 180
iparam.license_pause_time, 180
iparam.license_suppress_expire_wrns, 180
iparam.license_trh_expiry_wrn, 180
iparam.license_wait, 180
iparam.log, 180
iparam.log_ana_pro, 181
iparam.log_bi, 181
iparam.log_bi_freq, 181
iparam.log_check_convexity, 181
iparam.log_cut_second_opt, 181
iparam.log_expand, 181
iparam.log_factor, 182
iparam.log_feas_repair, 182
iparam.log_file, 182
iparam.log_head, 182
iparam.log_infeas_ana, 182
iparam.log_intpnt, 182
iparam.log_mio, 182
iparam.log_mio_freq, 182
iparam.log_optimizer, 183
iparam.log_order, 183
iparam.log_presolve, 183
iparam.log_response, 183
iparam.log_sensitivity, 183
iparam.log_sensitivity_opt, 183
iparam.log_sim, 183
iparam.log_sim_freq, 184
iparam.log_sim_minor, 184
iparam.log_storage, 184
iparam.max_num_warnings, 184
iparam.mio_branch_dir, 184
iparam.mio_construct_sol, 184
iparam.mio_cut_clique, 184
iparam.mio_cut_cmir, 185
iparam.mio_cut_gmi, 185
iparam.mio_cut_implied_bound, 185

www.EngineeringBooksPdf.com

iparam.mio_cut_knapsack_cover, 185
iparam.mio_cut_selection_level, 185
iparam.mio_heuristic_level, 185
iparam.mio_max_num_branches, 185
iparam.mio_max_num_relaxs, 186
iparam.mio_max_num_solutions, 186
iparam.mio_mode, 186
iparam.mio_mt_user_cb, 186
iparam.mio_node_optimizer, 186
iparam.mio_node_selection, 186
iparam.mio_perspective_reformulate, 186
iparam.mio_probing_level, 186
iparam.mio_rins_max_nodes, 187
iparam.mio_root_optimizer, 187
iparam.mio_root_repeat_presolve_level, 187
iparam.mio_vb_detection_level, 187
iparam.mt_spincount, 187
iparam.num_threads, 188
iparam.opf_max_terms_per_line, 188
iparam.opf_write_header, 188
iparam.opf_write_hints, 188
iparam.opf_write_parameters, 188
iparam.opf_write_problem, 188
iparam.opf_write_sol_bas, 188
iparam.opf_write_sol_itg, 189
iparam.opf_write_sol_itr, 189
iparam.opf_write_solutions, 188
iparam.optimizer, 189
iparam.param_read_case_name, 189
iparam.param_read_ign_error, 189
iparam.presolve_eliminator_max_fill, 189
iparam.presolve_eliminator_max_num_tries, 189
iparam.presolve_level, 189
iparam.presolve_lindep_abs_work_trh, 190
iparam.presolve_lindep_rel_work_trh, 190
iparam.presolve_lindep_use, 190
iparam.presolve_max_num_reductions, 190
iparam.presolve_use, 190
iparam.primal_repair_optimizer, 190
iparam.read_data_compressed, 190
iparam.read_data_format, 190
iparam.read_debug, 191
iparam.read_keep_free_con, 191
iparam.read_lp_drop_new_vars_in_bou, 191
iparam.read_lp_quoted_names, 191
iparam.read_mps_format, 191
iparam.read_mps_width, 191
iparam.read_task_ignore_param, 191
iparam.sensitivity_all, 191
iparam.sensitivity_optimizer, 192
iparam.sensitivity_type, 192
iparam.sim_basis_factor_use, 192
iparam.sim_degen, 192
iparam.sim_dual_crash, 192
iparam.sim_dual_phaseone_method, 192
iparam.sim_dual_restrict_selection, 192
iparam.sim_dual_selection, 193
iparam.sim_exploit_dupvec, 193
iparam.sim_hotstart, 193
iparam.sim_hotstart_lu, 193
iparam.sim_integer, 193
iparam.sim_max_iterations, 193
iparam.sim_max_num_setbacks, 193
iparam.sim_non_singular, 193
iparam.sim_primal_crash, 194
iparam.sim_primal_phaseone_method, 194
iparam.sim_primal_restrict_selection, 194
iparam.sim_primal_selection, 194
iparam.sim_refactor_freq, 194
iparam.sim_reformulation, 194
iparam.sim_save_lu, 194
iparam.sim_scaling, 195
iparam.sim_scaling_method, 195
iparam.sim_solve_form, 195
iparam.sim_stability_priority, 195
iparam.sim_switch_optimizer, 195
iparam.sol_filter_keep_basic, 195
iparam.sol_filter_keep_ranged, 195
iparam.sol_read_name_width, 196
iparam.sol_read_width, 196
iparam.solution_callback, 195
iparam.timing_level, 196
iparam.write_bas_constraints, 196
iparam.write_bas_head, 196
iparam.write_bas_variables, 196
iparam.write_data_compressed, 196
iparam.write_data_format, 197
iparam.write_data_param, 197
iparam.write_free_con, 197
iparam.write_generic_names, 197
iparam.write_generic_names_io, 197
iparam.write_ignore_incompatible_items, 197
iparam.write_int_constraints, 197
iparam.write_int_head, 197
iparam.write_int_variables, 197
iparam.write_lp_full_obj, 198
iparam.write_lp_line_width, 198
iparam.write_lp_quoted_names, 198
iparam.write_lp_strict_format, 198
iparam.write_lp_terms_per_line, 198
iparam.write_mps_format, 198
iparam.write_mps_int, 198
iparam.write_precision, 198
iparam.write_sol_barvariables, 199
iparam.write_sol_constraints, 199
iparam.write_sol_head, 199
iparam.write_sol_ignore_invalid_names, 199
iparam.write_sol_variables, 199
iparam.write_task_inc_sol, 199
iparam.write_xml_mode, 199
String params, 200
sparam.bas_sol_file_name, 200
sparam.data_file_name, 200
sparam.debug_file_name, 200
sparam.int_sol_file_name, 200

www.EngineeringBooksPdf.com

sparam.itr_sol_file_name, 200
sparam.mio_debug_string, 200
sparam.param_comment_sign, 200
sparam.param_read_file_name, 200
sparam.param_write_file_name, 200
sparam.read_mps_bou_name, 200
sparam.read_mps_obj_name, 201
sparam.read_mps_ran_name, 201
sparam.read_mps_rhs_name, 201
sparam.remote_access_token, 201
sparam.sensitivity_file_name, 201
sparam.sensitivity_res_file_name, 201
sparam.sol_filter_xc_low, 201
sparam.sol_filter_xc_upr, 201
sparam.sol_filter_xx_low, 201
sparam.sol_filter_xx_upr, 201
sparam.stat_file_name, 202
sparam.stat_key, 202
sparam.stat_name, 202
sparam.write_lp_gen_var_name, 202

Response codes

rescode.ok *(ok)*, 214
rescode.trm_internal *(trm_internal)*, 214
rescode.trm_internal_stop *(trm_internal_stop)*, 214
rescode.trm_max_iterations *(trm_max_iterations)*, 214
rescode.trm_max_num_setbacks *(trm_max_num_setbacks)*, 214
rescode.trm_max_time *(trm_max_time)*, 214
rescode.trm_mio_near_abs_gap *(trm_mio_near_abs_gap)*, 214
rescode.trm_mio_near_rel_gap *(trm_mio_near_rel_gap)*, 214
rescode.trm_mio_num_branches *(trm_mio_num_branches)*, 214
rescode.trm_mio_num_relaxs *(trm_mio_num_relaxs)*, 214
rescode.trm_num_max_num_int_solutions *(trm_num_max_num_int_solutions)*, 214
rescode.trm_numerical_problem *(trm_numerical_problem)*, 214
rescode.trm_objective_range *(trm_objective_range)*, 214
rescode.trm_stall *(trm_stall)*, 214
rescode.trm_user_callback *(trm_user_callback)*, 214
rescode.wrn_ana_almost_int_bounds *(wrn_ana_almost_int_bounds)*, 232
rescode.wrn_ana_c_zero *(wrn_ana_c_zero)*, 232
rescode.wrn_ana_close_bounds *(wrn_ana_close_bounds)*, 232
rescode.wrn_ana_empty_cols *(wrn_ana_empty_cols)*, 232
rescode.wrn_ana_large_bounds *(wrn_ana_large_bounds)*, 232
rescode.wrn_construct_invalid_sol_itg *(wrn_construct_invalid_sol_itg)*, 232
rescode.wrn_construct_no_sol_itg *(wrn_construct_no_sol_itg)*, 232
rescode.wrn_construct_solution_infeas *(wrn_construct_solution_infeas)*, 232
rescode.wrn_dropped_nz_qobj *(wrn_dropped_nz_qobj)*, 232
rescode.wrn_duplicate_barvariable_names *(wrn_duplicate_barvariable_names)*, 232
rescode.wrn_duplicate_cone_names *(wrn_duplicate_cone_names)*, 232
rescode.wrn_duplicate_constraint_names *(wrn_duplicate_constraint_names)*, 232
rescode.wrn_duplicate_variable_names *(wrn_duplicate_variable_names)*, 232
rescode.wrn_eliminator_space *(wrn_eliminator_space)*, 232
rescode.wrn_empty_name *(wrn_empty_name)*, 232
rescode.wrn_ignore_integer *(wrn_ignore_integer)*, 232
rescode.wrn_incomplete_linear_dependency_check *(wrn_incomplete_linear_dependency_check)*, 232
rescode.wrn_large_aij *(wrn_large_aij)*, 232
rescode.wrn_large_bound *(wrn_large_bound)*, 233
rescode.wrn_large_cj *(wrn_large_cj)*, 233
rescode.wrn_large_con_fx *(wrn_large_con_fx)*, 233
rescode.wrn_large_lo_bound *(wrn_large_lo_bound)*, 233
rescode.wrn_large_up_bound *(wrn_large_up_bound)*, 233
rescode.wrn_license_expire *(wrn_license_expire)*, 233
rescode.wrn_license_feature_expire *(wrn_license_feature_expire)*, 233
rescode.wrn_license_server *(wrn_license_server)*, 233
rescode.wrn_lp_drop_variable *(wrn_lp_drop_variable)*, 233
rescode.wrn_lp_old_quad_format *(wrn_lp_old_quad_format)*, 233
rescode.wrn_mio_infeasible_final *(wrn_mio_infeasible_final)*, 233
rescode.wrn_mps_split_bou_vector *(wrn_mps_split_bou_vector)*, 233
rescode.wrn_mps_split_ran_vector *(wrn_mps_split_ran_vector)*, 233
rescode.wrn_mps_split_rhs_vector *(wrn_mps_split_rhs_vector)*, 233
rescode.wrn_name_max_len *(wrn_name_max_len)*, 233

www.EngineeringBooksPdf.com

rescode.wrn_no_dualizer *(wrn_no_dualizer)*, 233
rescode.wrn_no_global_optimizer *(wrn_no_global_optimizer)*, 233
rescode.wrn_no_nonlinear_function_write *(wrn_no_nonlinear_function_write)*, 233
rescode.wrn_nz_in_upr_tri *(wrn_nz_in_upr_tri)*, 233
rescode.wrn_open_param_file *(wrn_open_param_file)*, 233
rescode.wrn_param_ignored_cmio *(wrn_param_ignored_cmio)*, 233
rescode.wrn_param_name_dou *(wrn_param_name_dou)*, 234
rescode.wrn_param_name_int *(wrn_param_name_int)*, 234
rescode.wrn_param_name_str *(wrn_param_name_str)*, 234
rescode.wrn_param_str_value *(wrn_param_str_value)*, 234
rescode.wrn_presolve_outofspace *(wrn_presolve_outofspace)*, 234
rescode.wrn_quad_cones_with_root_fixed_at_zero *(wrn_quad_cones_with_root_fixed_at_zero)*, 234
rescode.wrn_rquad_cones_with_root_fixed_at_zero *(wrn_rquad_cones_with_root_fixed_at_zero)*, 234
rescode.wrn_sol_file_ignored_con *(wrn_sol_file_ignored_con)*, 234
rescode.wrn_sol_file_ignored_var *(wrn_sol_file_ignored_var)*, 234
rescode.wrn_sol_filter *(wrn_sol_filter)*, 234
rescode.wrn_spar_max_len *(wrn_spar_max_len)*, 234
rescode.wrn_sym_mat_large *(wrn_sym_mat_large)*, 234
rescode.wrn_too_few_basis_vars *(wrn_too_few_basis_vars)*, 234
rescode.wrn_too_many_basis_vars *(wrn_too_many_basis_vars)*, 234
rescode.wrn_undef_sol_file_name *(wrn_undef_sol_file_name)*, 234
rescode.wrn_using_generic_names *(wrn_using_generic_names)*, 234
rescode.wrn_write_changed_names *(wrn_write_changed_names)*, 234
rescode.wrn_write_discarded_cfix *(wrn_write_discarded_cfix)*, 234
rescode.wrn_zero_aij *(wrn_zero_aij)*, 234
rescode.wrn_zeros_in_sparse_col *(wrn_zeros_in_sparse_col)*, 234
rescode.wrn_zeros_in_sparse_row *(wrn_zeros_in_sparse_row)*, 235
rescode.err_ad_invalid_codelist *(err_ad_invalid_codelist)*, 215
rescode.err_api_array_too_small *(err_api_array_too_small)*, 215
rescode.err_api_cb_connect *(err_api_cb_connect)*, 215
rescode.err_api_fatal_error *(err_api_fatal_error)*, 215
rescode.err_api_internal *(err_api_internal)*, 215
rescode.err_arg_is_too_large *(err_arg_is_too_large)*, 215
rescode.err_arg_is_too_small *(err_arg_is_too_small)*, 215
rescode.err_argument_dimension *(err_argument_dimension)*, 215
rescode.err_argument_is_too_large *(err_argument_is_too_large)*, 215
rescode.err_argument_lenneq *(err_argument_lenneq)*, 215
rescode.err_argument_perm_array *(err_argument_perm_array)*, 215
rescode.err_argument_type *(err_argument_type)*, 215
rescode.err_bar_var_dim *(err_bar_var_dim)*, 215
rescode.err_basis *(err_basis)*, 215
rescode.err_basis_factor *(err_basis_factor)*, 215
rescode.err_basis_singular *(err_basis_singular)*, 215
rescode.err_blank_name *(err_blank_name)*, 215
rescode.err_cannot_clone_nl *(err_cannot_clone_nl)*, 215
rescode.err_cannot_handle_nl *(err_cannot_handle_nl)*, 215
rescode.err_cbf_duplicate_acoord *(err_cbf_duplicate_acoord)*, 215
rescode.err_cbf_duplicate_bcoord *(err_cbf_duplicate_bcoord)*, 215
rescode.err_cbf_duplicate_con *(err_cbf_duplicate_con)*, 215
rescode.err_cbf_duplicate_int *(err_cbf_duplicate_int)*, 215
rescode.err_cbf_duplicate_obj *(err_cbf_duplicate_obj)*, 216
rescode.err_cbf_duplicate_objacoord *(err_cbf_duplicate_objacoord)*, 216
rescode.err_cbf_duplicate_var *(err_cbf_duplicate_var)*, 216
rescode.err_cbf_invalid_con_type *(err_cbf_invalid_con_type)*, 216
rescode.err_cbf_invalid_domain_dimension *(err_cbf_invalid_domain_dimension)*, 216
rescode.err_cbf_invalid_int_index *(err_cbf_invalid_int_index)*, 216
rescode.err_cbf_invalid_var_type *(err_cbf_invalid_var_type)*, 216

www.EngineeringBooksPdf.com

rescode.err_cbf_no_variables
(*err_cbf_no_variables*), 216
rescode.err_cbf_no_version_specified
(*err_cbf_no_version_specified*), 216
rescode.err_cbf_obj_sense
(*err_cbf_obj_sense*), 216
rescode.err_cbf_parse (*err_cbf_parse*), 216
rescode.err_cbf_syntax (*err_cbf_syntax*), 216
rescode.err_cbf_too_few_constraints
(*err_cbf_too_few_constraints*), 216
rescode.err_cbf_too_few_ints
(*err_cbf_too_few_ints*), 216
rescode.err_cbf_too_few_variables
(*err_cbf_too_few_variables*), 216
rescode.err_cbf_too_many_constraints
(*err_cbf_too_many_constraints*), 216
rescode.err_cbf_too_many_ints
(*err_cbf_too_many_ints*), 216
rescode.err_cbf_too_many_variables
(*err_cbf_too_many_variables*), 216
rescode.err_cbf_unsupported
(*err_cbf_unsupported*), 216
rescode.err_con_q_not_nsd
(*err_con_q_not_nsd*), 216
rescode.err_con_q_not_psd
(*err_con_q_not_psd*), 216
rescode.err_cone_index (*err_cone_index*), 217
rescode.err_cone_overlap (*err_cone_overlap*), 217
rescode.err_cone_overlap_append
(*err_cone_overlap_append*), 217
rescode.err_cone_rep_var
(*err_cone_rep_var*), 217
rescode.err_cone_size (*err_cone_size*), 217
rescode.err_cone_type (*err_cone_type*), 217
rescode.err_cone_type_str
(*err_cone_type_str*), 217
rescode.err_data_file_ext
(*err_data_file_ext*), 217
rescode.err_dup_name (*err_dup_name*), 217
rescode.err_duplicate_aij
(*err_duplicate_aij*), 217
rescode.err_duplicate_barvariable_names
(*err_duplicate_barvariable_names*), 217
rescode.err_duplicate_cone_names
(*err_duplicate_cone_names*), 217
rescode.err_duplicate_constraint_names
(*err_duplicate_constraint_names*), 217
rescode.err_duplicate_variable_names
(*err_duplicate_variable_names*), 217
rescode.err_end_of_file (*err_end_of_file*), 217
rescode.err_factor (*err_factor*), 217
rescode.err_feasrepair_cannot_relax
(*err_feasrepair_cannot_relax*), 217
rescode.err_feasrepair_inconsistent_bound
(*err_feasrepair_inconsistent_bound*), 217
rescode.err_feasrepair_solving_relaxed
(*err_feasrepair_solving_relaxed*), 217
rescode.err_file_license (*err_file_license*), 217
rescode.err_file_open (*err_file_open*), 217
rescode.err_file_read (*err_file_read*), 218
rescode.err_file_write (*err_file_write*), 218
rescode.err_first (*err_first*), 218
rescode.err_firsti (*err_firsti*), 218
rescode.err_firstj (*err_firstj*), 218
rescode.err_fixed_bound_values
(*err_fixed_bound_values*), 218
rescode.err_flexlm (*err_flexlm*), 218
rescode.err_global_inv_conic_problem
(*err_global_inv_conic_problem*), 218
rescode.err_huge_aij (*err_huge_aij*), 218
rescode.err_huge_c (*err_huge_c*), 218
rescode.err_identical_tasks
(*err_identical_tasks*), 218
rescode.err_in_argument (*err_in_argument*), 218
rescode.err_index (*err_index*), 218
rescode.err_index_arr_is_too_large
(*err_index_arr_is_too_large*), 218
rescode.err_index_arr_is_too_small
(*err_index_arr_is_too_small*), 218
rescode.err_index_is_too_large
(*err_index_is_too_large*), 218
rescode.err_index_is_too_small
(*err_index_is_too_small*), 218
rescode.err_inf_dou_index
(*err_inf_dou_index*), 218
rescode.err_inf_dou_name
(*err_inf_dou_name*), 218
rescode.err_inf_int_index
(*err_inf_int_index*), 218
rescode.err_inf_int_name
(*err_inf_int_name*), 218
rescode.err_inf_lint_index
(*err_inf_lint_index*), 218
rescode.err_inf_lint_name
(*err_inf_lint_name*), 218
rescode.err_inf_type (*err_inf_type*), 219
rescode.err_infeas_undefined
(*err_infeas_undefined*), 219
rescode.err_infinite_bound
(*err_infinite_bound*), 219
rescode.err_int64_to_int32_cast
(*err_int64_to_int32_cast*), 219
rescode.err_internal (*err_internal*), 219
rescode.err_internal_test_failed
(*err_internal_test_failed*), 219
rescode.err_inv_aptre (*err_inv_aptre*), 219
rescode.err_inv_bk (*err_inv_bk*), 219
rescode.err_inv_bkc (*err_inv_bkc*), 219
rescode.err_inv_bkx (*err_inv_bkx*), 219
rescode.err_inv_cone_type
(*err_inv_cone_type*), 219

www.EngineeringBooksPdf.com

rescode.err_inv_cone_type_str (err_inv_cone_type_str), 219
rescode.err_inv_marki (err_inv_marki), 219
rescode.err_inv_markj (err_inv_markj), 219
rescode.err_inv_name_item (err_inv_name_item), 219
rescode.err_inv_numi (err_inv_numi), 219
rescode.err_inv_numj (err_inv_numj), 219
rescode.err_inv_optimizer (err_inv_optimizer), 219
rescode.err_inv_problem (err_inv_problem), 219
rescode.err_inv_qcon_subi (err_inv_qcon_subi), 219
rescode.err_inv_qcon_subj (err_inv_qcon_subj), 219
rescode.err_inv_qcon_subk (err_inv_qcon_subk), 219
rescode.err_inv_qcon_val (err_inv_qcon_val), 219
rescode.err_inv_qobj_subi (err_inv_qobj_subi), 220
rescode.err_inv_qobj_subj (err_inv_qobj_subj), 220
rescode.err_inv_qobj_val (err_inv_qobj_val), 220
rescode.err_inv_sk (err_inv_sk), 220
rescode.err_inv_sk_str (err_inv_sk_str), 220
rescode.err_inv_skc (err_inv_skc), 220
rescode.err_inv_skn (err_inv_skn), 220
rescode.err_inv_skx (err_inv_skx), 220
rescode.err_inv_var_type (err_inv_var_type), 220
rescode.err_invalid_accmode (err_invalid_accmode), 220
rescode.err_invalid_aij (err_invalid_aij), 220
rescode.err_invalid_ampl_stub (err_invalid_ampl_stub), 220
rescode.err_invalid_barvar_name (err_invalid_barvar_name), 220
rescode.err_invalid_compression (err_invalid_compression), 220
rescode.err_invalid_con_name (err_invalid_con_name), 220
rescode.err_invalid_cone_name (err_invalid_cone_name), 220
rescode.err_invalid_file_format_for_cones (err_invalid_file_format_for_cones), 220
rescode.err_invalid_file_format_for_general_nl (err_invalid_file_format_for_general_nl), 220
rescode.err_invalid_file_format_for_sym_mat (err_invalid_file_format_for_sym_mat), 220
rescode.err_invalid_file_name (err_invalid_file_name), 220
rescode.err_invalid_format_type (err_invalid_format_type), 220
rescode.err_invalid_idx (err_invalid_idx), 220
rescode.err_invalid_iomode (err_invalid_iomode), 220
rescode.err_invalid_max_num (err_invalid_max_num), 221
rescode.err_invalid_name_in_sol_file (err_invalid_name_in_sol_file), 221
rescode.err_invalid_obj_name (err_invalid_obj_name), 221
rescode.err_invalid_objective_sense (err_invalid_objective_sense), 221
rescode.err_invalid_problem_type (err_invalid_problem_type), 221
rescode.err_invalid_sol_file_name (err_invalid_sol_file_name), 221
rescode.err_invalid_stream (err_invalid_stream), 221
rescode.err_invalid_surplus (err_invalid_surplus), 221
rescode.err_invalid_sym_mat_dim (err_invalid_sym_mat_dim), 221
rescode.err_invalid_task (err_invalid_task), 221
rescode.err_invalid_utf8 (err_invalid_utf8), 221
rescode.err_invalid_var_name (err_invalid_var_name), 221
rescode.err_invalid_wchar (err_invalid_wchar), 221
rescode.err_invalid_whichsol (err_invalid_whichsol), 221
rescode.err_json_data (err_json_data), 221
rescode.err_json_format (err_json_format), 221
rescode.err_json_missing_data (err_json_missing_data), 221
rescode.err_json_number_overflow (err_json_number_overflow), 221
rescode.err_json_string (err_json_string), 221
rescode.err_json_syntax (err_json_syntax), 221
rescode.err_last (err_last), 221
rescode.err_lasti (err_lasti), 221
rescode.err_lastj (err_lastj), 221
rescode.err_lau_arg_k (err_lau_arg_k), 222
rescode.err_lau_arg_m (err_lau_arg_m), 222
rescode.err_lau_arg_n (err_lau_arg_n), 222
rescode.err_lau_arg_trans (err_lau_arg_trans), 222
rescode.err_lau_arg_transa (err_lau_arg_transa), 222
rescode.err_lau_arg_transb (err_lau_arg_transb), 222

www.EngineeringBooksPdf.com

rescode.err_lau_arg_uplo *(err_lau_arg_uplo)*, 222
rescode.err_lau_invalid_lower_triangular_matrix *(err_lau_invalid_lower_triangular_matrix)*, 222
rescode.err_lau_invalid_sparse_symmetric_matrix *(err_lau_invalid_sparse_symmetric_matrix)*, 222
rescode.err_lau_not_positive_definite *(err_lau_not_positive_definite)*, 222
rescode.err_lau_singular_matrix *(err_lau_singular_matrix)*, 222
rescode.err_lau_unknown *(err_lau_unknown)*, 222
rescode.err_license *(err_license)*, 222
rescode.err_license_cannot_allocate *(err_license_cannot_allocate)*, 222
rescode.err_license_cannot_connect *(err_license_cannot_connect)*, 222
rescode.err_license_expired *(err_license_expired)*, 222
rescode.err_license_feature *(err_license_feature)*, 222
rescode.err_license_invalid_hostid *(err_license_invalid_hostid)*, 222
rescode.err_license_max *(err_license_max)*, 222
rescode. *(err_license_moseklm_daemon)*, 222
rescode.err_license_no_server_line *(err_license_no_server_line)*, 222
rescode.err_license_no_server_support *(err_license_no_server_support)*, 222
rescode.err_license_server *(err_license_server)*, 223
rescode.err_license_server_version *(err_license_server_version)*, 223
rescode.err_license_version *(err_license_version)*, 223
rescode.err_link_file_dll *(err_link_file_dll)*, 223
rescode.err_living_tasks *(err_living_tasks)*, 223
rescode.err_lower_bound_is_a_nan *(err_lower_bound_is_a_nan)*, 223
rescode.err_lp_dup_slack_name *(err_lp_dup_slack_name)*, 223
rescode.err_lp_empty *(err_lp_empty)*, 223
rescode.err_lp_file_format *(err_lp_file_format)*, 223
rescode.err_lp_format *(err_lp_format)*, 223
rescode.err_lp_free_constraint *(err_lp_free_constraint)*, 223
rescode.err_lp_incompatible *(err_lp_incompatible)*, 223
rescode.err_lp_invalid_con_name *(err_lp_invalid_con_name)*, 223
rescode.err_lp_invalid_var_name *(err_lp_invalid_var_name)*, 223
rescode.err_lp_write_conic_problem *(err_lp_write_conic_problem)*, 223
rescode.err_lp_write_geco_problem *(err_lp_write_geco_problem)*, 223
rescode.err_lu_max_num_tries *(err_lu_max_num_tries)*, 223
rescode.err_max_len_is_too_small *(err_max_len_is_too_small)*, 223
rescode.err_maxnumbarvar *(err_maxnumbarvar)*, 223
rescode.err_maxnumcon *(err_maxnumcon)*, 224
rescode.err_maxnumcone *(err_maxnumcone)*, 224
rescode.err_maxnumqnz *(err_maxnumqnz)*, 224
rescode.err_maxnumvar *(err_maxnumvar)*, 224
rescode.err_mio_internal *(err_mio_internal)*, 224
rescode.err_mio_invalid_node_optimizer *(err_mio_invalid_node_optimizer)*, 224
rescode.err_mio_invalid_root_optimizer *(err_mio_invalid_root_optimizer)*, 224
rescode.err_mio_no_optimizer *(err_mio_no_optimizer)*, 224
rescode.err_mio_not_loaded *(err_mio_not_loaded)*, 224
rescode.err_missing_license_file *(err_missing_license_file)*, 224
rescode.err_mixed_conic_and_nl *(err_mixed_conic_and_nl)*, 224
rescode.err_mps_cone_overlap *(err_mps_cone_overlap)*, 224
rescode.err_mps_cone_repeat *(err_mps_cone_repeat)*, 224
rescode.err_mps_cone_type *(err_mps_cone_type)*, 224
rescode.err_mps_duplicate_q_element *(err_mps_duplicate_q_element)*, 224
rescode.err_mps_file *(err_mps_file)*, 224
rescode.err_mps_inv_bound_key *(err_mps_inv_bound_key)*, 224
rescode.err_mps_inv_con_key *(err_mps_inv_con_key)*, 224
rescode.err_mps_inv_field *(err_mps_inv_field)*, 224
rescode.err_mps_inv_marker *(err_mps_inv_marker)*, 224
rescode.err_mps_inv_sec_name *(err_mps_inv_sec_name)*, 224
rescode.err_mps_inv_sec_order *(err_mps_inv_sec_order)*, 224
rescode.err_mps_invalid_obj_name *(err_mps_invalid_obj_name)*, 225
rescode.err_mps_invalid_objsense *(err_mps_invalid_objsense)*, 225
rescode.err_mps_mul_con_name *(err_mps_mul_con_name)*, 225
rescode.err_mps_mul_csec *(err_mps_mul_csec)*, 225

www.EngineeringBooksPdf.com

rescode.err_mps_mul_qobj
(*err_mps_mul_qobj*), 225
rescode.err_mps_mul_qsec
(*err_mps_mul_qsec*), 225
rescode.err_mps_no_objective
(*err_mps_no_objective*), 225
rescode.err_mps_non_symmetric_q
(*err_mps_non_symmetric_q*), 225
rescode.err_mps_null_con_name
(*err_mps_null_con_name*), 225
rescode.err_mps_null_var_name
(*err_mps_null_var_name*), 225
rescode.err_mps_splitted_var
(*err_mps_splitted_var*), 225
rescode.err_mps_tab_in_field2
(*err_mps_tab_in_field2*), 225
rescode.err_mps_tab_in_field3
(*err_mps_tab_in_field3*), 225
rescode.err_mps_tab_in_field5
(*err_mps_tab_in_field5*), 225
rescode.err_mps_undef_con_name
(*err_mps_undef_con_name*), 225
rescode.err_mps_undef_var_name
(*err_mps_undef_var_name*), 225
rescode.err_mul_a_element
(*err_mul_a_element*), 225
rescode.err_name_is_null
(*err_name_is_null*), 225
rescode.err_name_max_len
(*err_name_max_len*), 225
rescode.err_nan_in_blc (*err_nan_in_blc*), 225
rescode.err_nan_in_blx (*err_nan_in_blx*), 225
rescode.err_nan_in_buc (*err_nan_in_buc*), 225
rescode.err_nan_in_bux (*err_nan_in_bux*), 225
rescode.err_nan_in_c (*err_nan_in_c*), 226
rescode.err_nan_in_double_data
(*err_nan_in_double_data*), 226
rescode.err_negative_append
(*err_negative_append*), 226
rescode.err_negative_surplus
(*err_negative_surplus*), 226
rescode.err_newer_dll (*err_newer_dll*), 226
rescode.err_no_bars_for_solution
(*err_no_bars_for_solution*), 226
rescode.err_no_barx_for_solution
(*err_no_barx_for_solution*), 226
rescode.err_no_basis_sol (*err_no_basis_sol*), 226
rescode.err_no_dual_for_itg_sol
(*err_no_dual_for_itg_sol*), 226
rescode.err_no_dual_infeas_cer
(*err_no_dual_infeas_cer*), 226
rescode.err_no_init_env (*err_no_init_env*), 226
rescode.err_no_optimizer_var_type
(*err_no_optimizer_var_type*), 226
rescode.err_no_primal_infeas_cer
(*err_no_primal_infeas_cer*), 226
rescode.err_no_snx_for_bas_sol
(*err_no_snx_for_bas_sol*), 226
rescode.err_no_solution_in_callback
(*err_no_solution_in_callback*), 226
rescode.err_non_unique_array
(*err_non_unique_array*), 226
rescode.err_nonconvex (*err_nonconvex*), 226
rescode.err_nonlinear_equality
(*err_nonlinear_equality*), 226
rescode.err_nonlinear_functions_not_allowed
(*err_nonlinear_functions_not_allowed*), 226
rescode.err_nonlinear_ranged
(*err_nonlinear_ranged*), 226
rescode.err_nr_arguments
(*err_nr_arguments*), 226
rescode.err_null_env (*err_null_env*), 226
rescode.err_null_pointer (*err_null_pointer*), 227
rescode.err_null_task (*err_null_task*), 227
rescode.err_numconlim (*err_numconlim*), 227
rescode.err_numvarlim (*err_numvarlim*), 227
rescode.err_obj_q_not_nsd
(*err_obj_q_not_nsd*), 227
rescode.err_obj_q_not_psd
(*err_obj_q_not_psd*), 227
rescode.err_objective_range
(*err_objective_range*), 227
rescode.err_older_dll (*err_older_dll*), 227
rescode.err_open_dl (*err_open_dl*), 227
rescode.err_opf_format (*err_opf_format*), 227
rescode.err_opf_new_variable
(*err_opf_new_variable*), 227
rescode.err_opf_premature_eof
(*err_opf_premature_eof*), 227
rescode.err_optimizer_license
(*err_optimizer_license*), 227
rescode.err_overflow (*err_overflow*), 227
rescode.err_param_index (*err_param_index*), 227
rescode.err_param_is_too_large
(*err_param_is_too_large*), 227
rescode.err_param_is_too_small
(*err_param_is_too_small*), 227
rescode.err_param_name (*err_param_name*), 227
rescode.err_param_name_dou
(*err_param_name_dou*), 227
rescode.err_param_name_int
(*err_param_name_int*), 227
rescode.err_param_name_str
(*err_param_name_str*), 227
rescode.err_param_type (*err_param_type*), 228
rescode.err_param_value_str
(*err_param_value_str*), 228

www.EngineeringBooksPdf.com

rescode.err_platform_not_licensed (*err_platform_not_licensed*), 228
rescode.err_postsolve (*err_postsolve*), 228
rescode.err_pro_item (*err_pro_item*), 228
rescode.err_prob_license (*err_prob_license*), 228
rescode.err_qcon_subi_too_large (*err_qcon_subi_too_large*), 228
rescode.err_qcon_subi_too_small (*err_qcon_subi_too_small*), 228
rescode.err_qcon_upper_triangle (*err_qcon_upper_triangle*), 228
rescode.err_qobj_upper_triangle (*err_qobj_upper_triangle*), 228
rescode.err_read_format (*err_read_format*), 228
rescode.err_read_lp_missing_end_tag (*err_read_lp_missing_end_tag*), 228
rescode.err_read_lp_nonexisting_name (*err_read_lp_nonexisting_name*), 228
rescode.err_remove_cone_variable (*err_remove_cone_variable*), 228
rescode.err_repair_invalid_problem (*err_repair_invalid_problem*), 228
rescode.err_repair_optimization_failed (*err_repair_optimization_failed*), 228
rescode.err_sen_bound_invalid_lo (*err_sen_bound_invalid_lo*), 228
rescode.err_sen_bound_invalid_up (*err_sen_bound_invalid_up*), 228
rescode.err_sen_format (*err_sen_format*), 228
rescode.err_sen_index_invalid (*err_sen_index_invalid*), 228
rescode.err_sen_index_range (*err_sen_index_range*), 228
rescode.err_sen_invalid_regexp (*err_sen_invalid_regexp*), 228
rescode.err_sen_numerical (*err_sen_numerical*), 229
rescode.err_sen_solution_status (*err_sen_solution_status*), 229
rescode.err_sen_undef_name (*err_sen_undef_name*), 229
rescode.err_sen_unhandled_problem_type (*err_sen_unhandled_problem_type*), 229
rescode.err_server_connect (*err_server_connect*), 229
rescode.err_server_protocol (*err_server_protocol*), 229
rescode.err_server_status (*err_server_status*), 229
rescode.err_server_token (*err_server_token*), 229
rescode.err_size_license (*err_size_license*), 229
rescode.err_size_license_con (*err_size_license_con*), 229
rescode.err_size_license_intvar (*err_size_license_intvar*), 229
rescode.err_size_license_numcores (*err_size_license_numcores*), 229
rescode.err_size_license_var (*err_size_license_var*), 229
rescode.err_sol_file_invalid_number (*err_sol_file_invalid_number*), 229
rescode.err_solitem (*err_solitem*), 229
rescode.err_solver_probtype (*err_solver_probtype*), 229
rescode.err_space (*err_space*), 229
rescode.err_space_leaking (*err_space_leaking*), 229
rescode.err_space_no_info (*err_space_no_info*), 229
rescode.err_sym_mat_duplicate (*err_sym_mat_duplicate*), 229
rescode.err_sym_mat_huge (*err_sym_mat_huge*), 229
rescode.err_sym_mat_invalid (*err_sym_mat_invalid*), 230
rescode.err_sym_mat_invalid_col_index (*err_sym_mat_invalid_col_index*), 230
rescode.err_sym_mat_invalid_row_index (*err_sym_mat_invalid_row_index*), 230
rescode.err_sym_mat_invalid_value (*err_sym_mat_invalid_value*), 230
rescode.err_sym_mat_not_lower_tringular (*err_sym_mat_not_lower_tringular*), 230
rescode.err_task_incompatible (*err_task_incompatible*), 230
rescode.err_task_invalid (*err_task_invalid*), 230
rescode.err_task_write (*err_task_write*), 230
rescode.err_thread_cond_init (*err_thread_cond_init*), 230
rescode.err_thread_create (*err_thread_create*), 230
rescode.err_thread_mutex_init (*err_thread_mutex_init*), 230
rescode.err_thread_mutex_lock (*err_thread_mutex_lock*), 230
rescode.err_thread_mutex_unlock (*err_thread_mutex_unlock*), 230
rescode.err_toconic_constr_not_conic (*err_toconic_constr_not_conic*), 230
rescode.err_toconic_constr_q_not_psd (*err_toconic_constr_q_not_psd*), 230
rescode.err_toconic_constraint_fx (*err_toconic_constraint_fx*), 230
rescode.err_toconic_constraint_ra (*err_toconic_constraint_ra*), 230

www.EngineeringBooksPdf.com

rescode.err_toconic_objective_not_psd (*err_toconic_objective_not_psd*), 230
rescode.err_too_small_max_num_nz (*err_too_small_max_num_nz*), 230
rescode.err_too_small_maxnumanz (*err_too_small_maxnumanz*), 230
rescode.err_unb_step_size (*err_unb_step_size*), 230
rescode.err_undef_solution (*err_undef_solution*), 230
rescode.err_undefined_objective_sense (*err_undefined_objective_sense*), 231
rescode.err_unhandled_solution_status (*err_unhandled_solution_status*), 231
rescode.err_unknown (*err_unknown*), 231
rescode.err_upper_bound_is_a_nan (*err_upper_bound_is_a_nan*), 231
rescode.err_upper_triangle (*err_upper_triangle*), 231
rescode.err_user_func_ret (*err_user_func_ret*), 231
rescode.err_user_func_ret_data (*err_user_func_ret_data*), 231
rescode.err_user_nlo_eval (*err_user_nlo_eval*), 231
rescode.err_user_nlo_eval_hessubi (*err_user_nlo_eval_hessubi*), 231
rescode.err_user_nlo_eval_hessubj (*err_user_nlo_eval_hessubj*), 231
rescode.err_user_nlo_func (*err_user_nlo_func*), 231
rescode.err_whichitem_not_allowed (*err_whichitem_not_allowed*), 231
rescode.err_whichsol (*err_whichsol*), 231
rescode.err_write_lp_format (*err_write_lp_format*), 231
rescode.err_write_lp_non_unique_name (*err_write_lp_non_unique_name*), 231
rescode.err_write_mps_invalid_name (*err_write_mps_invalid_name*), 231
rescode.err_write_opf_invalid_var_name (*err_write_opf_invalid_var_name*), 231
rescode.err_writing_file (*err_writing_file*), 231
rescode.err_xml_invalid_problem_type (*err_xml_invalid_problem_type*), 231
rescode.err_y_is_undefined (*err_y_is_undefined*), 232

Types

booleant, 261
dparam, 261
env_t, 261
int32t, 261
int64t, 261
iparam, 261
realt, 261
rescode, 262
sparam, 262
string_t, 261
task_t, 261
userhandle_t, 261
wchart, 261

www.EngineeringBooksPdf.com