



MOSEK Optimizer API for Java

Release 8.1.0.15(beta)

MOSEK ApS

2017

1	Introduction	1
1.1	Why the Optimizer API for Java?	2
2	Contact Information	3
3	License Agreement	5
4	Installation	7
4.1	Building Examples and Testing the Installation	8
5	Basic Tutorials	11
5.1	The Basics Tutorial	12
5.2	Linear Optimization	14
5.3	Conic Quadratic Optimization	22
5.4	Semidefinite Optimization	26
5.5	Quadratic Optimization	31
5.6	Integer Optimization	38
5.7	Optimizer Termination Handling	42
5.8	Problem Modification and Reoptimization	44
5.9	Solution Analysis	49
5.10	Solver Parameters	54
6	Nonlinear Tutorials	57
6.1	Separable Convex (SCopt) Interface	57
7	Advanced Tutorials	61
7.1	Progress and data callback	61
7.2	Solving Linear Systems Involving the Basis Matrix	64
7.3	Calling BLAS/LAPACK Routines from MOSEK	71
7.4	Computing a Sparse Cholesky Factorization	73
7.5	Converting a quadratically constrained problem to conic form	76
7.6	MOSEK OptServer	79
8	Guidelines	83
8.1	Deployment	83
8.2	Efficiency Considerations	83
8.3	The license system	85
9	Case Studies	87
9.1	Portfolio Optimization	87
10	Errors and Warnings	107
10.1	Warnings	107
10.2	Errors	107
11	Managing I/O	109

11.1	Stream I/O	109
11.2	File I/O	110
11.3	Verbosity	110
12	Problem Formulation and Solutions	113
12.1	Linear Optimization	113
12.2	Conic Quadratic Optimization	116
12.3	Semidefinite Optimization	118
12.4	Quadratic and Quadratically Constrained Optimization	120
12.5	General Convex Optimization	121
13	The Optimizers for Continuous Problems	123
13.1	Presolve	123
13.2	Using Multiple Threads in an Optimizer	125
13.3	Linear Optimization	126
13.4	Conic Optimization	133
13.5	Nonlinear Convex Optimization	137
14	The Optimizer for Mixed-integer Problems	139
14.1	The Mixed-integer Optimizer Overview	139
14.2	Relaxations and bounds	139
14.3	Termination Criterion	140
14.4	Speeding Up the Solution Process	141
14.5	Understanding Solution Quality	141
14.6	The Optimizer Log	142
15	Problem Analyzer	143
15.1	General Characteristics	144
15.2	Objective	145
15.3	Linear Constraints	145
15.4	Constraint and Variable Bounds	146
15.5	Quadratic Constraints	146
15.6	Conic Constraints	146
16	Analyzing Infeasible Problems	147
16.1	Example: Primal Infeasibility	147
16.2	Locating the cause of Primal Infeasibility	148
16.3	Locating the Cause of Dual Infeasibility	149
16.4	The Infeasibility Report	149
16.5	Theory Concerning Infeasible Problems	151
16.6	The Certificate of Primal Infeasibility	151
16.7	The certificate of dual infeasibility	152
17	Sensitivity Analysis	153
17.1	Sensitivity Analysis for Linear Problems	153
17.2	Sensitivity Analysis with MOSEK	159
18	API Reference	163
18.1	API Conventions	163
18.2	Functions grouped by topic	168
18.3	Class Env	174
18.4	Class Task	184
18.5	Exceptions	279
18.6	Parameters grouped by topic	280
18.7	Parameters (alphabetical list sorted by type)	291
18.8	Response codes	328
18.9	Enumerations	350
18.10	Class types	375
18.11	Nonlinear extensions	376

19 Supported File Formats	379
19.1 The LP File Format	380
19.2 The MPS File Format	385
19.3 The OPF Format	397
19.4 The CBF Format	406
19.5 The XML (OSiL) Format	421
19.6 The Task Format	421
19.7 The JSON Format	421
19.8 The Solution File Format	429
20 List of examples	431
21 Interface changes	433
21.1 Compatibility	433
21.2 Functions	433
21.3 Parameters	434
21.4 Constants	436
21.5 Response Codes	440
Bibliography	443
Symbol Index	445
Index	461

INTRODUCTION

The **MOSEK** Optimization Suite 8.1.0.15(beta) is a powerful software package capable of solving large-scale optimization problems of the following kind:

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

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

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

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

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

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

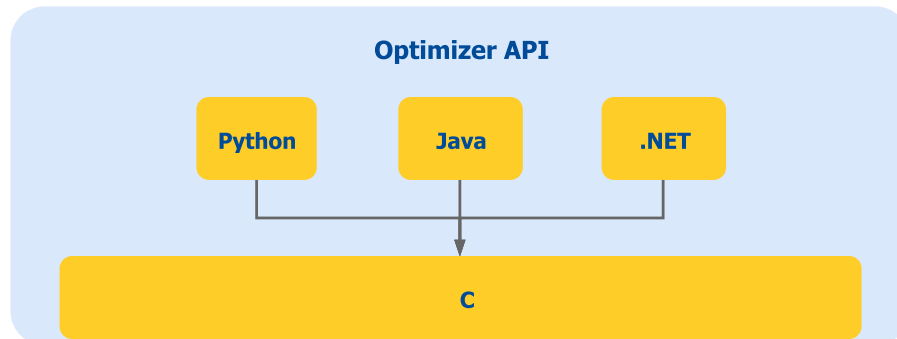
where $\mathcal{K} = \{y : y \geq 0\}$, i.e.,

$$\begin{aligned} Ax - b &= y, \\ y &\in \mathcal{K}. \end{aligned}$$

In conic optimization a wider class of convex sets \mathcal{K} is allowed, for example in 3 dimensions \mathcal{K} may correspond to an ice cream cone. The conic optimizer in **MOSEK** supports three structurally different types of cones \mathcal{K} , which allows a surprisingly large number of nonlinear relations to be modelled (as described in the [MOSEK modeling cookbook](#)), while preserving the nice algorithmic and theoretical properties of linear optimization.

1.1 Why the Optimizer API for Java?

The Optimizer API for Java 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 Java can be used with any application running on the Oracle Java platform (and possibly other Java implementations). It consists of a single class library `mosek.jar` and a set of library files that must be available at runtime.

The Optimizer API for Java provides access to:

- Linear Optimization (LO)
- Conic Quadratic (Second-Order Cone) Optimization (CQO, SOCO)
- Convex Quadratic and Quadratically Constrained Optimization (QCQO)
- Semidefinite Optimization (SDO)

as well as to additional functions for

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

CONTACT INFORMATION

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

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

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

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

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>.

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

zlib

MOSEK includes the *zlib* library obtained from the [zlib website](#). The license agreement for *zlib* is shown in [Listing 3.1](#).

Listing 3.1: *zlib* license.

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

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

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

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

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

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

fplib

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

Listing 3.2: *fplib* license.

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

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

INSTALLATION

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

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

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

Compatibility

The Optimizer API for Java is compatible with Java version 1.7 or later.

Locating Files

The files in Optimizer API for Java are organized as reported in [Table 4.1](#).

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

Relative Path	Description	Label
<MSKHOME>/mosek/8/tools/platform/<PLATFORM>/bin	Libraries and <i>jar</i> file	<JARDIR>
<MSKHOME>/mosek/8/tools/examples/java	Examples	<EXDIR>
<MSKHOME>/mosek/8/tools/examples/data	Additional data	<MISCDIR>

where

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

Setting up paths

To compile and run a Java program using **MOSEK** the correct path to <JARDIR>/mosek.jar must be provided in the Java classpath. This is usually set with the command line option

```
javac -d . -classpath <JARDIR>/mosek.jar lo1.java
java -classpath .:<JARDIR>/mosek.jar com.mosek.example.lo1
```

Alternatively, this can be set with the environment variable **CLASSPATH**. For more information about specifying class libraries and compiling applications, see the full Java documentation at <http://java.sun.com/>.

4.1 Building Examples and Testing the Installation

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

4.1.1 Windows

Building and executing a program

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

- Open a DOS prompt and go to the examples directory `<EXDIR>`.
- To compile a Java program and produce the class files in the current directory, type

```
javac -classpath <JARDIR>\mosek.jar -d . lo1.java
```

- To run the compiled program, type

```
java -classpath .:<JARDIR>\mosek.jar com.mosek.example.lo1
```

Compiling with Microsoft NMake

The distributed examples can also be compiled using Microsoft NMake. This requires that paths and environment is set up for Visual Studio tools (usually, the sub-menu containing Visual Studio also contains a Visual Studio *Command Prompt* which take care of all the necessary setup).

To build the examples, open a DOS box and change directory to the examples directory `<EXDIR>`. To compile all examples type

```
nmake /f Makefile
```

This will compile all the classes into a `jar` file. To run all the examples type

```
nmake /f Makefile test
```

4.1.2 Mac OS and Linux

Building and executing a program

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

- Open a console and go to the examples directory `<EXDIR>`.
- To compile a Java program and produce the class files in the current directory, type

```
javac -classpath <JARDIR>/mosek.jar -d . lo1.java
```

- To run the compiled program, type

```
java -classpath .:<JARDIR>/mosek.jar com.mosek.example.lo1
```

Compiling examples using make

The example directory contains makefiles for use with GNU Make. To build the examples, open a prompt and change directory to the examples directory <EXDIR>. To compile all examples type

```
make -f Makefile
```

This will compile all the classes into a `jar` file. To run all the examples type

```
make test
```


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 semidefinite 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

- 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.

5.1 The Basics Tutorial

The simplest program using the **MOSEK** Java 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.

5.1.1 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.

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

5.1.2 Example: Simple Working Example

The simple example in [Listing 5.1](#) shows a working Java 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 5.1: A simple code solving a problem loaded from file.

```
package com.mosek.example;
import mosek.*;

public class simple {
    public static void main (String[] args) {
        if (args.length == 0) {
            System.out.println ("Missing argument, syntax is:");
            System.out.println ("  simple inputfile [ solutionfile ]");
        } else {
            try (Env env = new Env();
                Task task = new Task(env, 0, 0)) {
```

```

task.set_Stream (mosek.streamtype.log,
new mosek.Stream() {
    public void stream(String msg) { System.out.print(msg); }
});
// We assume that a problem file was given as the first command
// line argument (received in `args')
task.readdata (args[0]);

// 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 (args.length >= 2) {
    // We define the output format to be OPF, and tell MOSEK to
    // leave out parameters and problem data from the output file.
    task.putintparam (mosek.iparam.write_data_format,    mosek.dataformat.op.value);
    task.putintparam (mosek.iparam.opf_write_solutions, mosek.onoffkey.on.value);
    task.putintparam (mosek.iparam.opf_write_hints,     mosek.onoffkey.off.value);
    task.putintparam (mosek.iparam.opf_write_parameters, mosek.onoffkey.off.value);
    task.putintparam (mosek.iparam.opf_write_problem,  mosek.onoffkey.off.value);
    task.writedata (args[1]);
}
}
}
}
}

```

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:

```
task.writedata (args[1]);
```

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

```
task.readdata (args[0]);
```

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.putdoublparam`, 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 5.10 for more details on how to set parameters.

A complete list of all parameters is found in Section 18.7.

5.2 Linear Optimization

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

Minimize or maximize the objective function

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

subject to the linear constraints

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

and the bounds

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

The problem description consists of the following elements:

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

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

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

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

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

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

5.2.1 Example LO1

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

$$\begin{aligned}
 &\text{maximize} && 3x_0 + 1x_1 + 5x_2 + 1x_3 \\
 &\text{subject to} && 3x_0 + 1x_1 + 2x_2 &= 30, \\
 &&& 2x_0 + 1x_1 + 3x_2 + 1x_3 &\geq 15, \\
 &&& & 2x_1 &+ 3x_3 &\leq 25,
 \end{aligned} \tag{5.1}$$

under the bounds

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

Solving the problem

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

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

Below we explain each of these steps.

Create an environment.

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

```
try (mosek.Env env = new Env();
```

Create an optimization task.

Next, an empty task object is created:

```
mosek.Task task = new Task(env, 0, 0) {
    // Directs the log task stream to the user specified
    // method task_msg_obj.stream
    task.set_Stream(
        mosek.streamtype.log,
        new mosek.Stream()
    { public void stream(String msg) { System.out.print(msg); } });
```

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

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

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

The bounds on variables are stored in the arrays

```
mosek.boundkey
bkg[] = {mosek.boundkey.lo,
        mosek.boundkey.ra,
        mosek.boundkey.lo,
        mosek.boundkey.lo
};
double blx[] = {0.0,
                0.0,
                0.0,
                0.0
};
double bux[] = {+infinity,
                10.0,
                +infinity,
                +infinity
};
```

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

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

The *Bound key* stored in **bkg** specify the type of the bound according to [Table 5.1](#).

Table 5.1: Bound keys as defined in the enum `boundkey`.

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

Interpretation of the bound keys.

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

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

and

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

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:

```
int    asub[][] = {
    {0, 1},
    {0, 1, 2},
    {0, 1},
    {1, 2}
};
double 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 5.3](#).

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

```
// Set the bounds on constraints.
// blc[i] <= constraint_i <= buc[i]
for (int i = 0; i < numcon; ++i)
    task.putconbound(i, bkc[i], blc[i], buc[i]);
```

Optimization

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

```
task.optimize();
```

Extracting the solution.

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

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

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

Catching exceptions

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

```
catch (mosek.Exception e) {  
    System.out.println ("An error/warning was encountered");  
    System.out.println (e.toString());  
    throw e;  
}
```

The types of exceptions that **MOSEK** can throw can be seen in [18.8](#).

Source code for lo1

Listing 5.2: Linear optimization example: complete listing.

```
package com.mosek.example;  
import mosek.*;  
  
public class lo1 {  
    static final int numcon = 3;  
    static final int numvar = 4;  
  
    public static void main (String[] args) {  
        // Since the value of infinity is ignored, we define it solely  
        // for symbolic purposes  
        double infinity = 0;  
  
        double c[] = {3.0, 1.0, 5.0, 1.0};  
        int asub[][] = {  
            {0, 1},  
            {0, 1, 2},  
            {0, 1},  
            {1, 2}  
        };  
        double aval[][] = {  
            {3.0, 2.0},  
            {1.0, 1.0, 2.0},  
            {2.0, 3.0},  
            {1.0, 3.0}  
        };  
        mosek.boundkey[]  
        bkc = {mosek.boundkey.fx,  
            mosek.boundkey.lo,  
            mosek.boundkey.up  
        };  
        double blc[] = {30.0,  
            15.0,  
            -infinity  
        };  
        double buc[] = {30.0,  
            +infinity,  
            25.0  
        };  
        mosek.boundkey  
        bkc[] = {mosek.boundkey.lo,  
            mosek.boundkey.ra,  
            mosek.boundkey.lo,  
            mosek.boundkey.lo  
        };  
        double blx[] = {0.0,
```



```

        0.0,
        0.0,
        0.0
    };
    double bux[] = { +infinity,
        10.0,
        +infinity,
        +infinity
    };
    double[] xx = new double[numvar];

    try (mosek.Env env = new Env();
        mosek.Task task = new Task(env, 0, 0)) {
        // Directs the log task stream to the user specified
        // method task_msg_obj.stream
        task.set_Stream(
            mosek.streamtype.log,
            new mosek.Stream()
        {
            public void stream(String msg) { System.out.print(msg); }
        });

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

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

        for (int j = 0; j < numvar; ++j) {
            // Set the linear term c_j in the objective.
            task.putcj(j, c[j]);

            // Set the bounds on variable j.
            // blx[j] <= x_j <= bux[j]
            task.putvarbound(j, bkg[j], blx[j], bux[j]);

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

        // Set the bounds on constraints.
        // blc[i] <= constraint_i <= buc[i]
        for (int i = 0; i < numcon; ++i)
            task.putconbound(i, bkc[i], blc[i], buc[i]);

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

        // Solve the problem
        task.optimize();

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

        // Get status information about the solution
        mosek.solsta solsta[] = new mosek.solsta[1];
        task.getsolsta(mosek.soltype.bas, solsta);

        switch (solsta[0]) {
            case optimal:

```

```

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

            System.out.println("Optimal primal solution\n");
            for (int j = 0; j < numvar; ++j)
                System.out.println ("x[" + j + "]: " + xx[j]);
            break;
        case dual_infeas_cer:
        case prim_infeas_cer:
        case near_dual_infeas_cer:
        case near_prim_infeas_cer:
            System.out.println("Primal or dual infeasibility certificate found.\n");
            break;
        case unknown:
            System.out.println("Unknown solution status.\n");
            break;
        default:
            System.out.println("Other solution status");
            break;
    }
}
}
catch (mosek.Exception e) {
    System.out.println ("An error/warning was encountered");
    System.out.println (e.toString());
    throw e;
}
}
}

```

Row-wise input

In the previous example the A matrix is set one column at a time. Alternatively the same matrix can be set one row at a time or the two methods can be mixed as in the example in Section 5.8. The following example show how to set the A matrix by rows

Listing 5.3: Example showing how to input the A matrix row-wise.

```

package com.mosek.example;
import mosek.*;

public class lo2 {
    static final int numcon = 3;
    static final int numvar = 4;
    static final int NUMANZ = 9;

    public static void main (String[] args) {
        // Since the value infinity is never used, we define
        // 'infinity' symbolic purposes only
        double
        infinity = 0;

        double c[] = {3.0, 1.0, 5.0, 1.0};
        int asub[][] = {
            {0, 1, 2},
            {0, 1, 2, 3},
            {1, 3}
        };
        double aval[][] = {
            {3.0, 1.0, 2.0 },
            {2.0, 1.0, 3.0, 1.0},

```

```

    {2.0, 3.0}
};
mosek.boundkey[]
bkc    = {mosek.boundkey.fx,
          mosek.boundkey.lo,
          mosek.boundkey.up
        };
double blc[] = {30.0,
                15.0,
                -infinity
        };
double buc[] = {30.0,
                +infinity,
                25.0
        };

mosek.boundkey
bkc[] = {mosek.boundkey.lo,
        mosek.boundkey.ra,
        mosek.boundkey.lo,
        mosek.boundkey.lo
        };
double blx[] = {0.0,
                0.0,
                0.0,
                0.0
        };
double bux[] = {+infinity,
                10.0,
                +infinity,
                +infinity
        };
double[] xx = new double[numvar];

try (Env env = new Env();
     Task task = new Task(env, 0, 0)) {
    // Directs the log task stream to the user specified
    // method task_msg_obj.stream
    task.set_Stream(
        mosek.streamtype.log,
        new mosek.Stream()
    {
        public void stream(String msg) { System.out.print(msg); }
    });

    /* Give MOSEK an estimate of the size of the input data.
       This is done to increase the speed of inputting data.
       However, it is optional. */
    /* Append 'numcon' empty constraints.
       The constraints will initially have no bounds. */
    task.appendcons(numcon);

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

    for (int j = 0; j < numvar; ++j) {
        /* Set the linear term c_j in the objective.*/
        task.putcj(j, c[j]);
        /* Set the bounds on variable j.
           blx[j] <= x_j <= bux[j] */
        task.putbound(mosek.accmode.var, j, bkc[j], blx[j], bux[j]);
    }
    /* Set the bounds on constraints.
       for i=1, ..., numcon : blc[i] <= constraint i <= buc[i] */
    for (int i = 0; i < numcon; ++i) {

```

```

    task.putbound(mosek.accmode.con, i, bkc[i], blc[i], buc[i]);

    /* 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. */
}

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

/* Solve the problem */
mosek.rescode r = task.optimize();

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

mosek.solsta solsta[] = new mosek.solsta[1];
/* Get status information about the solution */
task.getsolsta(mosek.soltype.bas, solsta);

task.getxx(mosek.soltype.bas, // Basic solution.
           xx);
switch (solsta[0]) {
case optimal:
case near_optimal:
    System.out.println("Optimal primal solution\n");
    for (int j = 0; j < numvar; ++j)
        System.out.println ("x[" + j + "]: " + xx[j]);
    break;
case dual_infeas_cer:
case prim_infeas_cer:
case near_dual_infeas_cer:
case near_prim_infeas_cer:
    System.out.println("Primal or dual infeasibility.\n");
    break;
case unknown:
    System.out.println("Unknown solution status.\n");
    break;
default:
    System.out.println("Other solution status");
    break;
}
}
catch (mosek.Exception e) {
    System.out.println ("An error/warning was encountered");
    System.out.println (e.toString());
    throw e;
}
}
}

```

5.3 Conic Quadratic Optimization

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

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

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

a product of smaller cones and x is the full problem variable.

MOSEK can solve conic quadratic optimization problems of the form

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

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

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

For convenience, a user defining a conic quadratic problem only needs to specify subsets of variables x^t belonging to quadratic cones. These are:

- 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:

$$\mathcal{Q}_r^n = \left\{ x \in \mathbb{R}^n : 2x_0x_1 \geq \sum_{j=2}^{n-1} x_j^2, \quad x_0 \geq 0, \quad x_1 \geq 0 \right\}.$$

For example, the following constraint:

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

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

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

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

5.3.1 Example CQO1

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

$$\begin{aligned} & \text{minimize} && x_4 + x_5 + x_6 \\ & \text{subject to} && x_1 + x_2 + 2x_3 = 1, \\ & && x_1, x_2, x_3 \geq 0, \\ & && x_4 \geq \sqrt{x_1^2 + x_2^2}, \\ & && 2x_5x_6 \geq x_3^2 \end{aligned} \tag{5.2}$$

Implementation

Problem (5.2) can be implemented using the **MOSEK** Java API as follows:

Listing 5.4: Source code solving problem (5.2).

```

package com.mosek.example;

import mosek.*;

public class cq01 {
    static final int numcon = 1;
    static final int numvar = 6;

    public static void main (String[] args) throws java.lang.Exception {
        // Since the value infinity is never used, we define
        // 'infinity' symbolic purposes only
        double infinity = 0;

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

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

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

        double[][] aval = {
            {1.0},
            {1.0},
            {2.0}
        };
        int[][] asub = {
            {0},
            {0},
            {0}
        };

        int[] csub = new int[3];
    }
}

```

```

double[] xx = new double[numvar];
// create a new environment object
try (Env env = new Env();
     Task task = new Task(env, 0, 0)) {
    // Directs the log task stream to the user specified
    // method task_msg_obj.stream
    task.set_Stream(
        mosek.streamtype.log,
        new mosek.Stream()
    {
        public void stream(String msg) { System.out.print(msg); }
    });

    /* Give MOSEK an estimate of the size of the input data.
       This is done to increase the speed of inputting data.
       However, it is optional. */
    /* Append 'numcon' empty constraints.
       The constraints will initially have no bounds. */
    task.appendcons(numcon);

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

    /* Optionally add a constant term to the objective. */
    task.putcfix(0.0);
    for (int j = 0; j < numvar; ++j) {
        /* Set the linear term c_j in the objective.*/
        task.putcj(j, c[j]);
        /* Set the bounds on variable j.
           blx[j] <= x_j <= bux[j] */
        task.putbound(mosek.accmode.var, j, bkg[j], blx[j], bux[j]);
    }

    for (int j = 0; j < aval.length; ++j)
        /* Input column j of A */
        task.putacol(j,
                     a[j], /* Variable (column) index.*/
                     asub[j], /* Row index of non-zeros in column j.*/
                     aval[j]); /* Non-zero Values of column j. */

    /* Set the bounds on constraints.
       for i=1, ..., numcon : blc[i] <= constraint i <= buc[i] */
    for (int i = 0; i < numcon; ++i)
        task.putbound(mosek.accmode.con, i, bkc[i], blc[i], buc[i]);

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

    csub[0] = 4;
    csub[1] = 5;
    csub[2] = 2;

    task.appendcone(mosek.conetype.rquad, 0.0, csub);

    task.putobjsense(mosek.objsense.minimize);

    System.out.println ("optimize");
    /* Solve the problem */
    mosek.rescode r = task.optimize();
    System.out.println (" Mosek warning:" + r.toString());
    // Print a summary containing information
    // about the solution for debugging purposes

```

```

task.solutionsummary(mosek.streamtype.msg);

mosek.solsta solsta[] = new mosek.solsta[1];

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

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

switch (solsta[0]) {
case optimal:
case near_optimal:
    System.out.println("Optimal primal solution\n");
    for (int j = 0; j < numvar; ++j)
        System.out.println ("x[" + j + "]: " + xx[j]);
    break;
case dual_infeas_cer:
case prim_infeas_cer:
case near_dual_infeas_cer:
case near_prim_infeas_cer:
    System.out.println("Primal or dual infeasibility.\n");
    break;
case unknown:
    System.out.println("Unknown solution status.\n");
    break;
default:
    System.out.println("Other solution status");
    break;
}
} catch (mosek.Exception e) {
    System.out.println ("An error/warning was encountered");
    System.out.println (e.toString());
    throw e;
}
}
}

```

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

```

task.appendcone(mosek.conetype.quad,
               0.0, /* For future use only, can be set to 0.0 */
               csub);

```

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.

5.4 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

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

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

MOSEK can solve semidefinite optimization problems of the form

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

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

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

5.4.1 Example SDO1

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

$$\begin{aligned} & \text{minimize} && \left\langle \begin{bmatrix} 2 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 2 \end{bmatrix}, \bar{X} \right\rangle + x_0 \\ & \text{subject to} && \left\langle \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \bar{X} \right\rangle + x_0 = 1, \\ & && \left\langle \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \bar{X} \right\rangle + x_1 + x_2 = 1/2, \\ & && x_0 \geq \sqrt{x_1^2 + x_2^2}, \quad \bar{X} \succeq 0, \end{aligned} \tag{5.3}$$

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

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

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

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

subject to the two linear constraints

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

Problem (5.3) is implemented in Listing 5.5.

Listing 5.5: Source code solving problem (5.3).

```
package com.mosek.example;
import mosek.*;

public class sdo1 {
    public static void main(String[] argv) {
        int    numcon      = 2; /* Number of constraints. */
        int    numvar      = 3; /* Number of conic quadratic variables */
        int    numanz      = 3; /* Number of non-zeros in A */
        int    numbarvar    = 1; /* Number of semidefinite variables */
        int    dimbarvar[] = {3}; /* Dimension of semidefinite cone */
```

```

int    lenbarvar[] = {3 * (3 + 1) / 2}; /* Number of scalar SD variables */

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

double[] blc = { 1.0, 0.5 };
double[] buc = { 1.0, 0.5 };

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

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

int[][]  bara_i = { {0, 1, 2}, {0, 1, 2, 1, 2, 2} },
        bara_j = { {0, 1, 2}, {0, 0, 0, 1, 1, 2} };

double[][] bara_v = { {1.0, 1.0, 1.0}, {1.0, 1.0, 1.0, 1.0, 1.0, 1.0} };
int[]     conesub = { 0, 1, 2 };

try (Env env = new Env();
     Task task = new Task(env, 0, 0)) {
    // Directs the log task stream to the user specified
    // method task_msg_obj.stream
    task.set_Stream(
        mosek.streamtype.log,
        new mosek.Stream()
    {
        public void stream(String msg) { System.out.print(msg); }
    });

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

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

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

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

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

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

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

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

    /* Set the bounds on constraints.

```

```

    for i=1, ..., numcon : blc[i] <= constraint i <= buc[i] */

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

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

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

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

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

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

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

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

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

mosek.solsta[] solsta = new mosek.solsta[1];
task.getsolsta (mosek.soltype.itr, solsta);

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

        task.getxx(mosek.soltype.itr, xx);
        task.getbarxj(mosek.soltype.itr, /* Request the interior solution. */

```

```

        0,
        barx);
    System.out.println("Optimal primal solution");
    for (int i = 0; i < numvar; ++i)
        System.out.println("x[" + i + "] : " + xx[i]);

    for (int i = 0; i < lenbarvar[0]; ++i)
        System.out.println("barx[" + i + "]: " + barx[i]);
    break;
case dual_infeas_cer:
case prim_infeas_cer:
case near_dual_infeas_cer:
case near_prim_infeas_cer:
    System.out.println("Primal or dual infeasibility certificate found.");
    break;
case unknown:
    System.out.println("The status of the solution could not be determined.");
    break;
default:
    System.out.println("Other solution status.");
    break;
}
}
}
}

```

Source code comments

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

```
task.appendbarvars(dimbarvar);
```

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

```

idx[0] = task.appendsparsesymmat(dimbarvar[0],
                                barc_i,
                                barc_j,
                                barc_v);

```

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 \bar{C}_j using *Task.putbarcj*, which forms a linear combination of one more symmetric matrices:

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

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 \bar{A}_{ij} is setup by the function *Task.putbaraij*:

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

where the second argument specifies the constraint number (the corresponding row of \bar{A}), and the third argument specifies the semidefinite variable index (the corresponding column of \bar{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,    /* Request the interior solution. */
             0,
             barx);
```

The function returns the half-vectorization of \bar{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.

5.5 Quadratic Optimization

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

$$\begin{aligned} & \text{minimize} && \frac{1}{2}x^T Q^o x + c^T x + c^f \\ & \text{subject to} && \begin{aligned} l_k^c &\leq \frac{1}{2}x^T Q^k x + \sum_{j=0}^{n-1} a_{k,j} x_j &\leq u_k^c, & k = 0, \dots, m-1, \\ l_j^x &\leq x_j &\leq u_j^x, & j = 0, \dots, n-1. \end{aligned} \end{aligned} \quad (5.4)$$

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 k th constraint must be of the form

$$l_k^c \leq \frac{1}{2}x^T Q^k x + \sum_{j=0}^{n-1} a_{k,j} x_j \quad (5.5)$$

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.

5.5.1 Example: Quadratic Objective

$$\begin{aligned} & \text{minimize} && x_1^2 + 0.1x_2^2 + x_3^2 - x_1x_3 - x_2 \\ & \text{subject to} && 1 \leq x_1 + x_2 + x_3 \\ & && x \geq 0. \end{aligned} \tag{5.6}$$

For the example (5.6) 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 (5.4) which implies that diagonal elements must be doubled in Q , i.e. $Q_{11} = 2$, whereas the coefficient in (5.6) 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 5.6.

Listing 5.6: Source code implementing problem (5.6).

```
package com.mosek.example;
import mosek.*;

public class qo1 {
    static final int numcon = 1;    /* Number of constraints. */
    static final int numvar = 3;    /* Number of variables. */
    static final int NUMANZ = 3;    /* Number of numzeros in A. */
    static final int NUMQNZ = 4;    /* Number of nonzeros in Q. */

    public static void main (String[] args) {
        // Since the value infinity is never used, we define
        // 'infinity' symbolic purposes only
        double infinity = 0;
        double[] c = {0.0, -1.0, 0.0};

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

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

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

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

```

double[] xx    = new double[numvar];

try (Env env = new Env();
     Task task = new Task(env, 0, 0)) {
    // Directs the log task stream to the user specified
    // method task_msg_obj.stream
    task.set_Stream(
        mosek.streamtype.log,
        new mosek.Stream()
    {
        public void stream(String msg) { System.out.print(msg); }
    });
    /* Give MOSEK an estimate of the size of the input data.
       This is done to increase the speed of inputting data.
       However, it is optional. */
    /* Append 'numcon' empty constraints.
       The constraints will initially have no bounds. */
    task.appendcons(numcon);

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

    for (int j = 0; j < numvar; ++j) {
        /* Set the linear term c_j in the objective.*/
        task.putcj(j, c[j]);
        /* Set the bounds on variable j.
           blx[j] <= x_j <= bux[j] */
        task.putbound(mosek.accmode.var, j, bkg[j], blx[j], bux[j]);
        /* Input column j of A */
        task.putacol(j,
                     asub[j],      /* Variable (column) index.*/
                     aval[j]);     /* Row index of non-zeros in column j.*/
    }
    /* Set the bounds on constraints.
       for i=1, ..., numcon : blc[i] <= constraint i <= buc[i] */
    for (int i = 0; i < numcon; ++i)
        task.putbound(mosek.accmode.con, i, bkc[i], blc[i], buc[i]);

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

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

    /* Input the Q for the objective. */

    task.putqobj(qsubi, qsubj, qval);

    /* Solve the problem */
    mosek.rescode r = task.optimize();
    System.out.println (" Mosek warning:" + r.toString());
    // Print a summary containing information
    // about the solution for debugging purposes
    task.solutionsummary(mosek.streamtype.msg);

    mosek.solsta solsta[] = new mosek.solsta[1];
    /* Get status information about the solution */
    task.getsolsta(mosek.soltype.itr, solsta);

    /* Get the solution */
    task.getxx(mosek.soltype.itr, // Interior solution.

```

```

        xx);

    switch (solsta[0]) {
    case optimal:
    case near_optimal:
        System.out.println("Optimal primal solution\n");
        for (int j = 0; j < numvar; ++j)
            System.out.println ("x[" + j + "]: " + xx[j]);
        break;
    case dual_infeas_cer:
    case prim_infeas_cer:
    case near_dual_infeas_cer:
    case near_prim_infeas_cer:
        System.out.println("Primal or dual infeasibility\n");
        break;
    case unknown:
        System.out.println("Unknown solution status.\n");
        break;
    default:
        System.out.println("Other solution status");
        break;
    }
}
}
catch (mosek.Exception e) {
    System.out.println ("An error/warning was encountered");
    System.out.println (e.toString());
    throw e;
}
} /* Main */
}

```

Example code comments

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

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

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

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

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

The lower triangular part of the matrix Q^o is specified using an unordered sparse triplet format (for details, see Section 18.1.4):

```

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

```

Please note that

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

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


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

5.5.2 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 (5.5).

Consider the problem:

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

This is equivalent to

$$\begin{aligned} & \text{minimize} && \frac{1}{2}x^T Q^o x + c^T x \\ & \text{subject to} && \frac{1}{2}x^T Q^0 x + Ax \geq b, \end{aligned} \tag{5.7}$$

where

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

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

Listing 5.7: Script implementing problem (5.7).

```
package com.mosek.example;
import mosek.*;

public class qcqol {
    static final int numcon = 1;    /* Number of constraints. */
    static final int numvar = 3;    /* Number of variables. */
    static final int NUMANZ = 3;    /* Number of numzeros in A. */
    static final int NUMQNZ = 4;    /* Number of nonzeros in Q. */

    public static void main (String[] args) {
        // Since the value infinity is never used, we define
        // 'infinity' symbolic purposes only
        double infinity = 0;
        double[] c = {0.0, -1.0, 0.0};

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

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

```

        infinity
    };

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

    double[] xx    = new double[numvar];

    try (mosek.Env env = new mosek.Env();
        mosek.Task task = new mosek.Task(env, 0, 0)) {
        // Directs the log task stream to the user specified
        // method task_msg_obj.stream
        task.set_Stream(
            mosek.streamtype.log,
            new mosek.Stream()
        { public void stream(String msg) { System.out.print(msg); } });

        /* Give MOSEK an estimate of the size of the input data.
        This is done to increase the speed of inputting data.
        However, it is optional. */
        /* Append 'numcon' empty constraints.
        The constraints will initially have no bounds. */
        task.appendcons(numcon);

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

        for (int j = 0; j < numvar; ++j) {
            /* Set the linear term c_j in the objective.*/
            task.putc(j, c[j]);
            /* Set the bounds on variable j.
            blx[j] <= x_j <= bux[j] */
            task.putbound(mosek.accmode.var, j, bxx[j], blx[j], bux[j]);
            /* Input column j of A */
            task.putacol(j,
                asub[j],          /* Variable (column) index.*/
                aval[j]);         /* Row index of non-zeros in column j.*/
                                   /* Non-zero Values of column j. */
        }

        /* Set the bounds on constraints.
        for i=1, ..., numcon : blc[i] <= constraint i <= buc[i] */
        for (int i = 0; i < numcon; ++i)
            task.putbound(mosek.accmode.con, i, bkc[i], blc[i], buc[i]);

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

        int[] qosubi = { 0, 1, 2, 2 };
        int[] qosubj = { 0, 1, 0, 2 };
        double[] qoal = { 2.0, 0.2, -1.0, 2.0 };

        /* Input the Q for the objective. */

        task.putqobj(qosubi, qosubj, qoal);

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

```

```

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

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

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

task.putobjsense(mosek.objsense.minimize);

/* Solve the problem */

try {
    mosek.rescode termcode = task.optimize();
} catch (mosek.Warning e) {
    System.out.println (" Mosek warning:");
    System.out.println (e.toString ());
}

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

mosek.solsta solsta[] = new mosek.solsta[1];
/* Get status information about the solution */
task.getsolsta(mosek.soltype.itr, solsta);

task.getxx(mosek.soltype.itr, // Interior solution.
           xx);
switch (solsta[0]) {
    case optimal:
    case near_optimal:
        System.out.println("Optimal primal solution\n");
        for (int j = 0; j < numvar; ++j)
            System.out.println ("x[" + j + "]:" + xx[j]);
        break;
    case dual_infeas_cer:
    case prim_infeas_cer:
    case near_dual_infeas_cer:
    case near_prim_infeas_cer:
        System.out.println("Primal or dual infeasibility.\n");
        break;
    case unknown:
        System.out.println("Unknown solution status.\n");
        break;
    default:
        System.out.println("Other solution status");
        break;
}

}

catch (mosek.Exception e) {
    System.out.println ("An error/warning was encountered");
    System.out.println (e.msg);
    throw e;
}

} /* Main */
}

```

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

5.6 Integer Optimization

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

5.6.1 Example MILO1

We use the example

$$\begin{aligned} &\text{maximize} && x_0 + 0.64x_1 \\ &\text{subject to} && 50x_0 + 31x_1 \leq 250, \\ & && 3x_0 - 2x_1 \geq -4, \\ & && x_0, x_1 \geq 0 \quad \text{and integer} \end{aligned} \tag{5.8}$$

to demonstrate how to set up and solve a problem with integer variables. It has the structure of a linear optimization problem (see 5.2) except for integrality constraints on the variables. Therefore, only the specification of the integer constraints requires something new compared to the linear optimization problem discussed previously.

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

```
for (int j = 0; j < numvar; ++j)
    task.putvartype(j, mosek.variabletype.type_int);
```

The complete source for the example is listed Listing 5.8. 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 5.8: Source code implementing problem (5.8).

```
package com.mosek.example;
import mosek.*;

public class milo1 {
    static final int numcon = 2;
    static final int numvar = 2;

    public static void main (String[] args) {
        // Since the value infinity is never used, we define
        // 'infinity' symbolic purposes only
        double infinity = 0;

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

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

        double[] c = { 1.0, 0.64 };
    }
}
```

```

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

int[] ptrb = { 0, 2 };
int[] ptre = { 2, 4 };

double[] xx = new double[numvar];

try (Env env = new Env();
    Task task = new Task(env, 0, 0)) {
    // Directs the log task stream to the user specified
    // method task_msg_obj.stream
    task.set_Stream(
        mosek.streamtype.log,
        new mosek.Stream()
    {
        public void stream(String msg) { System.out.print(msg); });
    task.set_ItgSolutionCallback(
        new mosek.ItgSolutionCallback() {
            public void callback(double[] xx) {
                System.out.print("New integer solution: ");
                for (double v : xx) System.out.print(" " + v + " ");
                System.out.println("");
            }
        });
    /* Append 'numcon' empty constraints.
    The constraints will initially have no bounds. */
    task.appendcons(numcon);

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

    for (int j = 0; j < numvar; ++j) {
        /* Set the linear term c_j in the objective.*/
        task.putcj(j, c[j]);
        /* Set the bounds on variable j.
        blx[j] <= x_j <= bux[j] */
        task.putvarbound(j, bkc[j], blx[j], bux[j]);
        /* Input column j of A */
        task.putacol(j,
            asub[j], /* Variable (column) index.*/
            aval[j]); /* Row index of non-zeros in column j.*/
    }
    /* Set the bounds on constraints.
    for i=1, ..., numcon : blc[i] <= constraint i <= buc[i] */
    for (int i = 0; i < numcon; ++i)
        task.putconbound(i, bkc[i], blc[i], buc[i]);

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

    /* A maximization problem */
    task.putobjsense(mosek.objsense.maximize);
    /* Solve the problem */
    try {
        task.optimize();
    } catch (mosek.Warning e) {
        System.out.println(" Mosek warning:");
        System.out.println(e.toString());
    }
}

```

```

// Print a summary containing information
// about the solution for debugging purposes
task.solutionsummary(mosek.streamtype.msg);
task.getxx(mosek.soltype.itg, // Integer solution.
           xx);
mosek.solsta solsta[] = new mosek.solsta[1];
/* Get status information about the solution */
task.getsolsta(mosek.soltype.itg, solsta);

switch (solsta[0]) {
case integer_optimal:
case near_integer_optimal:
    System.out.println("Optimal solution\n");
    for (int j = 0; j < numvar; ++j)
        System.out.println ("x[" + j + "]: " + xx[j]);
    break;
case prim_feas:
    System.out.println("Feasible solution\n");
    for (int j = 0; j < numvar; ++j)
        System.out.println ("x[" + j + "]: " + xx[j]);
    break;

case unknown:
    mosek.prosta prosta[] = new mosek.prosta[1];
    task.getprosta(mosek.soltype.itg, prosta);
    switch (prosta[0]) {
        case prim_infeas_or_unbounded:
            System.out.println("Problem status Infeasible or unbounded");
            break;
        case prim_infeas:
            System.out.println("Problem status Infeasible.");
            break;
        case unknown:
            System.out.println("Problem status unknown.");
            break;
        default:
            System.out.println("Other problem status.");
            break;
    }
    break;
default:
    System.out.println("Other solution status");
    break;
}
}
catch (mosek.Exception e) {
    System.out.println ("An error or warning was encountered");
    System.out.println (e.getMessage ());
    throw e;
}
}
}

```

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 5.10 for more details on how to set solver parameters.

5.6.2 Specifying an initial solution

Solution time of 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, **MOSEK** will be forced to compute the remaining continuous variable values. If the specified integer solution is infeasible or incomplete, **MOSEK** will simply ignore it.

We concentrate on a simple example below.

$$\begin{aligned}
 &\text{maximize} && 7x_0 + 10x_1 + x_2 + 5x_3 \\
 &\text{subject to} && x_0 + x_1 + x_2 + x_3 \leq 2.5 \\
 &&& x_0, x_1, x_2 \in \mathbb{Z} \\
 &&& x_0, x_1, x_2, x_3 \geq 0
 \end{aligned} \tag{5.9}$$

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 5.9: Implementation of problem (5.9) specifying an initial solution.

```

package com.mosek.example;
import mosek.*;

class msgclass extends mosek.Stream {
    public msgclass () {
        super ();
    }

    public void stream (String msg) {
        System.out.print (msg);
    }
}

public class miointsol {
    public static void main (String[] args) {
        // Since the value infinity is never used, we define
        // 'infinity' symbolic purposes only
        double
        infinity = 0;

        int numvar = 4;
        int numcon = 1;

        double[] c = { 7.0, 10.0, 1.0, 5.0 };

        mosek.boundkey[] bkc = {mosek.boundkey.up};
        double[] blc = { -infinity};
        double[] buc = {2.5};
        mosek.boundkey[] bkc
        = {mosek.boundkey.lo,
          mosek.boundkey.lo,
          mosek.boundkey.lo,
          mosek.boundkey.lo
        };
        double[] blx = {0.0,
                       0.0,
                       0.0,
                       0.0
        };
        double[] bux = {infinity,
                       infinity,
                       infinity,
                       infinity
        };
    }
}

```

```

int[]    ptrb   = {0, 1, 2, 3};
int[]    ptre   = {1, 2, 3, 4};
double[] aval   = {1.0, 1.0, 1.0, 1.0};
int[]    asub   = {0, 0, 0, 0 };
int[]    intsub  = {0, 1, 2};
mosek.variabletype[] inttype = {mosek.variabletype.type_int,
                                mosek.variabletype.type_int,
                                mosek.variabletype.type_int
                                };
double[]  intxx  = {0.0, 2, 0, 0, 0};

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

    task.inputdata(numcon, numvar,
                  c, 0.0,
                  ptrb, ptre,
                  asub, aval,
                  bkc, blc, buc,
                  bkx, blx, bux);

    task.putvartypelist(intsub, inttype);

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

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

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

    // Assign values 1,1,0 to integer variables
    task.putxxslice(mosek.soltype.itg, 0, 3, intxx);

    // solve
    task.optimize();
} catch (mosek.Exception e)
/* Catch both Error and Warning */
{
    System.out.println ("An error was encountered");
    System.out.println (e.getMessage ());
    throw e;
}
}

```

5.7 Optimizer Termination Handling

After solving an optimization problem with **MOSEK** an appropriate 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 handling:

- **Termination code:** It provides information about why the optimizer terminated. For instance if a time limit has been specified (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 prespecified 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 18.8 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 5.10 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 5.10: A typical code that handle **MOSEK** response code.

```
package com.mosek.example;
import mosek.*;

public class response {
    public static void main(String[] argv) {
        if (argv.length < 1) {
            System.out.println("No input file specified");
        } else {

            try (Env env = new Env();
                 Task task = new Task(env, 0, 0)) {
                // Directs the log task stream to the user specified
                // method task_msg_obj.stream
                task.set_Stream(
                    streamtype.log,
                    new mosek.Stream()
                { public void stream(String msg) { System.out.print(msg); } });
            }
        }
    }
}
```

```

task.readdata(argv[0]);

rescode trm = task.optimize();

solsta[] solsta = new solsta[1]; task.getsolsta(soltype.itr, solsta);
switch ( solsta[0] ) {
  case optimal:
  case near_optimal:
    System.out.println("An optimal basic solution is located.");

    task.solutionsummary(streamtype.msg);
    break;
  case dual_infeas_cer:
  case near_dual_infeas_cer:
    System.out.println("Dual infeasibility certificate found.");
    break;
  case prim_infeas_cer:
  case near_prim_infeas_cer:
    System.out.println("Primal infeasibility certificate found.");
    break;
  case unknown: {
    StringBuffer symname = new StringBuffer();
    StringBuffer desc = new StringBuffer();

    /* The solutions status is unknown. The termination code
       indicating why the optimizer terminated prematurely. */

    System.out.println("The solution status is unknown.");
    /* No system failure e.g. an iteration limit is reached. */

    Env.getcodedesc(trm, symname, desc);
    System.out.printf("  Termination code: %s\n", symname);
    break;
  }
  default:
    System.out.println("An unexpected solution status is obtained.");
    break;
}
}
}
}
}
}

```

5.8 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 feasible, 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 17.

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

5.8.1 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:

$$\begin{aligned}
 &\text{maximize} && 1.5x_0 + 2.5x_1 + 3.0x_2 \\
 &\text{subject to} && 2x_0 + 4x_1 + 3x_2 \leq 100000, \\
 &&& 3x_0 + 2x_1 + 3x_2 \leq 50000, \\
 &&& 2x_0 + 3x_1 + 2x_2 \leq 60000,
 \end{aligned} \tag{5.10}$$

and

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

Code in Listing 5.11 loads and solves this problem:

Listing 5.11: How to load problem (5.10)

```

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

double c[]      = {1.5,
                   2.5,
                   3.0
                   };

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

double blc[]    = { -infinity,
                   -infinity,
                   -infinity
                   };

```

```

double buc[]          = {100000,
                          50000,
                          60000
                          };

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

double blc[]          = {0.0,
                          0.0,
                          0.0
                          };

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

int asub[][] = {
  {0, 1, 2},
  {0, 1, 2},
  {0, 1, 2}
};

double aval[][] = {
  { 2.0, 3.0, 2.0 },
  { 4.0, 2.0, 3.0 },
  { 3.0, 3.0, 2.0 }
};

double[] xx = new double[numvar];

try (Env env = new Env();
     Task task = new Task(env, 0, 0)) {
  /* Append the constraints. */
  task.appendcons(numcon);

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

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

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

  /* Put variable bounds. */
  for (int j = 0; j < numvar; ++j)
    task.putbound(mosek.accmode.var, j, bkc[j], blc[j], bux[j]);

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

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

```

```

/* Solve the problem */

termcode = task.optimize();

task.solutionsummary(mosek.streamtype.msg);

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

for (int j = 0; j < numvar; ++j)
    System.out.println ("x[" + j + "]: " + xx[j]);

```

5.8.2 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);
```

The problem now has the form:

$$\begin{aligned}
 &\text{maximize} && 1.5x_0 + 2.5x_1 + 3.0x_2 \\
 &\text{subject to} && 3x_0 + 4x_1 + 3x_2 \leq 100000, \\
 & && 3x_0 + 2x_1 + 3x_2 \leq 50000, \\
 & && 2x_0 + 3x_1 + 2x_2 \leq 60000,
 \end{aligned} \tag{5.11}$$

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

5.8.3 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 5.12](#)

Listing 5.12: How to add a column.

```

/* Get index of new variable. */
int[] varidx = new int[1];
task.getnumvar(varidx);

/* Append a new variable x_3 to the problem */
task.appendvars(1);

/* Set bounds on new variable */
task.putbound(mosek.acemode.var,
              varidx[0],
              mosek.boundkey.lo,
              0,
              +infinity);

/* Change objective */
task.putcj(varidx[0], 1.0);

```

```

/* Put new values in the A matrix */
int[] acolsub = new int[] {0, 2};
double[] acolval = new double[] {4.0, 1.0};

task.putacol(varidx[0], /* column index */
             acolsub,
             acolval);

```

After this operation the problem looks this way:

$$\begin{aligned}
 &\text{maximize} && 1.5x_0 + 2.5x_1 + 3.0x_2 + 1.0x_3 \\
 &\text{subject to} && 3x_0 + 4x_1 + 3x_2 + 4x_3 \leq 100000, \\
 & && 3x_0 + 2x_1 + 3x_2 \leq 50000, \\
 & && 2x_0 + 3x_1 + 2x_2 + 1x_3 \leq 60000,
 \end{aligned} \tag{5.12}$$

and

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

5.8.4 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.

In this case an optimal solution to problem (5.11) was found and then added a column was added to get (5.12). 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.value);
termcode = task.optimize();

```

5.8.5 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:

```

/* Get index of new constraint. */
int[] conidx = new int[1];
task.getnumcon(conidx);

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

/* Set bounds on new constraint */
task.putbound(

```

```

mosek.accmode.con,
conidx[0],
mosek.boundkey.up,
-infinity,
30000);

/* Put new values in the A matrix */

int[] arowsub = new int[] {0, 1, 2, 3 };
double[] arowval = new double[] {1.0, 2.0, 1.0, 1.0};

task.putarow(conidx[0], /* row index */
             arowsub,
             arowval);

```

5.9 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 5.9.1). The solution summary provides information about

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

By inspecting the solution summary it can be verified that **MOSEK** produces a feasible solution, and, in the continuous case, the optimality can be checked using the dual solution. Furthermore, the problem itself can be inspected using the problem analyzer discussed in Section 15.

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 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

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

and **MOSEK** reports the optimal solution

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

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

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

may have been inputted instead of

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

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

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

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

5.9.1 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 must 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 associated with each solution are reported.

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

```
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 interpretation 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 measures. 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 perturbations 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

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

if and only if a y exists such that

$$\begin{aligned} A^T y &\leq 0, \\ b^T y &> 0. \end{aligned} \tag{5.13}$$

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 (5.13). 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 positive and large compared to the violations. Note the Farkas certificate is a ray so any positive 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. Moreover, 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 5.13 shows how to use these function to determine the quality of the solution.

Listing 5.13: An example of solution quality analysis.

```
package com.mosek.example;
import mosek.*;
```

```

public class solutionquality {
    public static void main (String[] args) {
        if (args.length == 0) {
            System.out.println ("Missing argument, syntax is:");
            System.out.println ("  solutionquality inputfile");
        } else {
            try (Env env = new Env();
                 Task task = new Task(env, 0, 0)) {
                task.set_Stream (mosek.streamtype.log,
                                new mosek.Stream() {
                                    public void stream(String msg) { System.out.print(msg); }
                                });
                // We assume that a problem file was given as the first command
                // line argument (received in `args`)
                task.readdata (args[0]);

                // Solve the problem
                task.optimize ();

                // System.Out.Println (a summary of the solution
                task.solutionsummary (mosek.streamtype.log);

                mosek.solsta solsta[] = new mosek.solsta[1];
                task.getsolsta(mosek.soltype.bas, solsta);

                double pobj[] = new double[1];
                double pviolcon[] = new double[1];
                double pviolvar[] = new double[1];
                double pviolbarvar[] = new double[1];
                double pviolcones[] = new double[1];
                double pviolitg[] = new double[1];
                double dobj[] = new double[1];
                double dviolcon[] = new double[1];
                double dviolvar[] = new double[1];
                double dviolbarvar[] = new double[1];
                double dviolcones[] = new double[1];

                task.getsolutioninfo(mosek.soltype.bas,
                                     pobj, pviolcon, pviolvar, pviolbarvar, pviolcones, pviolitg,
                                     dobj, dviolcon, dviolvar, dviolbarvar, dviolcones);

                switch (solsta[0]) {
                    case optimal:
                    case near_optimal:

                        double abs_obj_gap    = Math.abs(dobj[0] - pobj[0]);
                        double rel_obj_gap    = abs_obj_gap / (1.0 + Math.min(Math.abs(pobj[0]), Math.
→abs(dobj[0])));
                        double max_primal_viol = Math.max(pviolcon[0], pviolvar[0]);
                        max_primal_viol = Math.max(max_primal_viol , pviolbarvar[0]);
                        max_primal_viol = Math.max(max_primal_viol , pviolcones[0]);

                        double max_dual_viol  = Math.max(dviolcon[0], dviolvar[0]);
                        max_dual_viol  = Math.max(max_dual_viol , dviolbarvar[0]);
                        max_dual_viol  = Math.max(max_dual_viol , dviolcones[0]);

                        // 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

                        System.out.println ("Customized solution information.\n");
                        System.out.println ("  Absolute objective gap: " + abs_obj_gap);

```

```

System.out.println (" Relative objective gap: " + rel_obj_gap);
System.out.println (" Max primal violation : " + max_primal_viol);
System.out.println (" Max dual violation : " + max_dual_viol);

boolean accepted = true;

if ( rel_obj_gap > 1e-6 ) {
    System.out.println ("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 ) {
    System.out.println ("Warning: Primal violation is too LARGE");
    accepted = false;
}

if (max_dual_viol > 1e-6 ) {
    System.out.println ("Warning: Dual violation is too LARGE.");
    accepted = false;
}

if ( accepted ) {
    int numvar = task.getnumvar();
    System.out.println ("Optimal primal solution");
    double xj[] = new double[1];
    for (int j = 0; j < numvar; j++) {
        task.getxxslice(mosek.soltype.bas, j, j + 1, xj);
        System.out.println ("x[" + j + "]: " + xj[0]);
    }
} else {
    // print etailed information about the solution
    task.analyzesolution(mosek.streamtype.log, mosek.soltype.bas);
}
break;

case dual_infeas_cer:
case prim_infeas_cer:
case near_dual_infeas_cer:
case near_prim_infeas_cer:
    System.out.println ("Primal or dual infeasibility certificate found.");
    break;
case unknown:
    System.out.println ("The status of the solution is unknown.");
    break;
default:
    System.out.println ("Other solution status");
}
} catch (mosek.Exception e) {
    System.out.println ("An error/warning was encountered");
    System.out.println (e.toString());
    throw e;
}
}
}
}

```

5.9.2 The Solution Summary for Mixed-Integer Problems

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

Listing 5.14: Example of solution summary for a mixed-integer problem.

```
Integer 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 difference compared to the continuous 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. Moreover, 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.

5.10 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 18.7.

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);
```

If the specified value is not valid, the API will rise an exception of type *rescode.err_param_value_str* and the parameter left unchanged.

```
try {
    task.putintparam(mosek.iparam.log, -1);
} catch (mosek.Error e) {
    System.out.println(" -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.value);
```

For more information about other parameter related functions, please browse the API reference in Section 18.

The complete code for this tutorial follows in Listing 5.15.

Listing 5.15: Parameter setting example.

```
package com.mosek.example;
import mosek.*;

public class parameters {

    public static void main (String[] args) {
        try {
            mosek.Env env = new Env();
            mosek.Task task = new Task(env, 0, 0);

            int param;

            System.out.println("Test MOSEK parameter get/set functions");

            param = task.getintparam(mosek.iparam.log);
            System.out.println("Default value for parameter task.ipar.log= " + param);

            System.out.println(" setting to 1 using putintparam...");
            task.putintparam(mosek.iparam.log, 1);

            System.out.println(" setting to -1 using putintparam...");
            try {
                task.putintparam(mosek.iparam.log, -1);
            } catch (mosek.Error e) {
                System.out.println(" -1 rejected as not a valid value");
            }

            System.out.println(" setting to 2 using putparam...");
            task.putparam("MSK_IPAR_LOG", "2");

            System.out.println(" setting to 3 using putnaintparam...");
            task.putnaintparam("MSK_IPAR_LOG", 3);

            System.out.println(" selecting the dual simplex algorithm...");
            task.putintparam( mosek.iparam.optimizer, mosek.optimizertype.dual_simplex.value);

        } catch (mosek.Exception e) {
            System.out.println ("An error/warning was encountered");
            System.out.println (e.toString());
            throw e;
        }
    }
}
```


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.

6.1 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 :

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

With the separable convex interface SCopt it is possible to add non-linear functions of the form

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

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 is supported; each w_k^i can be one of the following:

Table 6.1: Functions supported by the SCopt interface.

Separable function	Operator name	Name
$f x \ln(x)$	<i>ent</i>	Entropy function
$f e^{g x + h}$	<i>exp</i>	Exponential function
$f \ln(g x + h)$	<i>log</i>	Logarithm
$f(x + h)^g$	<i>pow</i>	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.

Some simple rules can be followed 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
$f x \ln(x)$	<i>ent</i>	$0 < x$.
$f e^{g x + h}$	<i>exp</i>	$-\infty < x < \infty$.
$f \ln(g x + h)$	<i>log</i>	If $g > 0$: $-h/g < x$.
		If $g < 0$: $x < -h/g$.
$f(x + h)^g$	<i>pow</i>	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 conditions for the relevant ranges for $f > 0$ — changing the sign of f switches concavity/convexity.

Separable function	Operator name	Convexity conditions
$f x \ln(x)$	<i>ent</i>	Convex within safe bounds.
$f e^{g x + h}$	<i>exp</i>	Convex for all x .
$f \ln(g x + h)$	<i>log</i>	Concave within safe bounds.
$f(x + h)^g$	<i>pow</i>	If g is even integer: convex within safe bounds.
		If g is odd integer: <ul style="list-style-type: none"> • concave if $(-\infty, -h)$, • convex if $(-h, \infty)$
		If $0 < g < 1$: concave within safe bounds.
		Otherwise: convex within safe bounds.

Important: Do NOT use the SCopt API unless you really have to. First try to reformulate the problem in conic form. The SCopt API is less stable, less predictable, harder to debug and has inferior status info.

A problem involving linear combinations of variables (for example under the logarithm function), can be converted to a separable problem using slack variables and additional constraints. The evaluation of the nonlinear functions, their gradients and Hessians is much easier in the separable case.

6.1.1 Example

Consider the following separable convex problem:

$$\begin{aligned}
 &\text{minimize} && \exp(x_2) - \ln(x_1) \\
 &\text{subject to} && x_2 \ln(x_2) \leq 0 \\
 & && x_1^{1/2} - x_2 \geq 0 \\
 & && \frac{1}{2} \leq x_1, x_2 \leq 1.
 \end{aligned} \tag{6.1}$$

Note that all nonlinear functions are well defined for x values satisfying the variable bounds strictly. This assures that function evaluation errors will not occur during the optimization process because **MOSEK**.

The linear part of the problem is specified as usually. The nonlinear part is set using the function `Task.putSCEval`. See the [API reference](#) for a description of the format. After that a standard invocation of `Task.optimize` solves the problem. The [API reference](#) describes additional functions for reading and writing SOpt terms from/to a file.

Listing 6.1: Implementation of problem (6.1).

```

package com.mosek.example;
import mosek.*;

public class scopt1 {
    public static void main(String[] args) {
        try (Env env = new Env();
             Task task = new Task(env, 0, 0)) {
            task.set_Stream(
                mosek.streamtype.log,
                new mosek.Stream()
                { public void stream(String msg) { System.out.print(msg); } });

            int numvar = 2;
            int numcon = 2;
            double inf = 0.;

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

            double[] blc = new double[] { -inf, .0 };
            double[] buc = new double[] { .0, inf };

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

            double[] blx = new double[] { 0.5, 0.5 };
            double[] bux = new double[] { 1.0, 1.0 };

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

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

            task.putaij(1, 1, -1.0);

            mosek.scopr[] opro = new mosek.scopr[] {mosek.scopr.log, mosek.scopr.exp};
            int[] oprjo = new int[] { 0, 1 };
            double[] oprfo = new double[] { -1.0, 1.0 };
            double[] oprgo = new double[] { 1.0, 1.0 };
        }
    }
}

```

```
double[] oprho      = new double[] { 0.0, 0.0 };

mosek.scopr[] oprc  = new mosek.scopr[] { mosek.scopr.ent, mosek.scopr.pow };
int[] opric         = new int[]      { 0, 1 };
int[] oprjc         = new int[]      { 1, 0 };
double[] oprfc       = new double[]  { 1.0, 1.0 };
double[] oprgc       = new double[]  { .0, 0.5 };
double[] oprhc       = new double[]  { .0, 0.0 };

task.putSCeval(opro, oprjo, oprfo, oprgo, oprho,
               oprc, opric, oprjc, oprfc, oprgc, oprhc);
task.putintparam(mosek.iparam.write_ignore_incompatible_items, 1);
task.writeSC("scopt1.sco", "scopt1.opf");

task.optimize();

double[] res = new double[numvar];
task.getsolutionslice(
    mosek.soltype.itr,
    mosek.solitem.xx,
    0, numvar,
    res);

System.out.print("Solution is: [ " + res[0]);
for (int i = 1; i < numvar; ++i) System.out.print(", " + res[i]);
System.out.println(" ]");
} catch (mosek.Exception e) {
    System.out.println ("An error/warning was encountered");
    System.out.println (e.toString());
    throw e;
}
}
```

ADVANCED TUTORIALS

7.1 Progress and data callback

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

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

Optimizer API for Java has the following callback mechanisms:

- **progress callback**, which provides only the basic status of the solver.
- **data callback**, which provides the solver status and a complete set of information items that describe the progress of the optimizer in detail.
- **integer solution callback**, for reporting progress on a mixed-integer problem.

Warning

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

Retrieving mixed-integer solutions

If the mixed-integer optimizer is used, the callback will take place, in particular, every time an improved integer solution is found. In that case it is possible to retrieve the current values of the best integer solution from within the callback function. It can be useful for implementing complex termination criteria for integer optimization. Note that there is a specialized callback class for retrieving only the integer solution anyway.

7.1.1 Data callback

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

The callback is set by calling the method `Task.set_InfoCallback`. The callback function must be implemented by extending the abstract class `DataCallback` and implementing the method `DataCallback.callback`.

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

7.1.2 Progress callback

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

The callback is set by calling the method `Task.set_Progress`. The callback function must be implemented by extending the abstract class `Progress` and implementing the method `Progress.progress`.

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

7.1.3 Integer solution callback

In this type of callback the user-defined callback function receives an updated solution every time the mixed-integer optimizer improves the objective value. It can be useful for implementing complex termination criteria for integer optimization.

Syntax

The callback is set by calling the method `Task.set_ItgSolutionCallback`. The callback function must be implemented by extending the abstract class `ItgSolutionCallback` and implementing the method `ItgSolutionCallback.callback`.

7.1.4 Working example: Data callback

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

Listing 7.1: An example of a data callback function.

```
private static DataCallback makeUserCallback(final double maxtime) {
    return new DataCallback() {
        public int callback(callbackcode caller,
                           double[]    douinf,
                           int[]       intinf,
                           long[]      lintinf) {
            double opttime = 0.0;
            int itrn;
            double pobj, dobj, stime;

            Formatter f = new Formatter(System.out);
            switch (caller) {
                case begin_intpnt:
                    f.format("Starting interior-point optimizer\n");
                    break;
                case intpnt:
                    itrn    = intinf[iinfitem.intpnt_iter.value];
                    pobj    = douinf[dinfitem.intpnt_primal_obj.value];
                    dobj    = douinf[dinfitem.intpnt_dual_obj.value];
                    stime   = douinf[dinfitem.intpnt_time.value];
                    opttime = douinf[dinfitem.optimizer_time.value];

                    f.format("Iterations: %-3d\n", itrn);
            }
        }
    };
}
```

```

        f.format("   Time: %6.2f(%6.2f) ", opttime, stime);
        f.format("   Primal obj.: %-18.6e   Dual obj.: %-18.6e\n", pobj, dobj);
        break;
    case end_intpnt:
        f.format("Interior-point optimizer finished.\n");
        break;
    case begin_primal_simplex:
        f.format("Primal simplex optimizer started.\n");
        break;
    case update_primal_simplex:
        itrn    = intinf[iinfitem.sim_primal_iter.value ];
        pobj    = douinf[dinfitem.sim_obj.value         ];
        stime   = douinf[dinfitem.sim_time.value        ];
        opttime = douinf[dinfitem.optimizer_time.value  ];

        f.format("Iterations: %-3d\n", itrn);
        f.format("   Elapsed time: %6.2f(%6.2f)\n", opttime, stime);
        f.format("   Obj.: %-18.6e", pobj );
        break;
    case end_primal_simplex:
        f.format("Primal simplex optimizer finished.\n");
        break;
    case begin_dual_simplex:
        f.format("Dual simplex optimizer started.\n");
        break;
    case update_dual_simplex:
        itrn    = intinf[iinfitem.sim_dual_iter.value   ];
        pobj    = douinf[dinfitem.sim_obj.value         ];
        stime   = douinf[dinfitem.sim_time.value        ];
        opttime = douinf[dinfitem.optimizer_time.value  ];
        f.format("Iterations: %-3d\n", itrn);
        f.format("   Elapsed time: %6.2f(%6.2f)\n", opttime, stime);
        f.format("   Obj.: %-18.6e\n", pobj);
        break;
    case end_dual_simplex:
        f.format("Dual simplex optimizer finished.\n");
        break;
    case begin_bi:
        f.format("Basis identification started.\n");
        break;
    case end_bi:
        f.format("Basis identification finished.\n");
        break;
    default:
    }
    System.out.flush();
    if (opttime >= maxtime)
        // mosek is spending too much time. Terminate it.
        return 1;

    return 0;
}
};
}

```

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

Listing 7.2: Attaching the data callback function to the model.

```
task.set_InfoCallback(makeUserCallback(maxtime));
```

7.2 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

$$B\bar{x} = w \tag{7.1}$$

and

$$B^T \bar{x} = w \tag{7.2}$$

each has a unique solution for all w .

MOSEK provides functions for solving the linear systems (7.1) and (7.2) for an arbitrary w .

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

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

7.2.1 Identifying the Basis

To use the solutions to (7.1) and (7.2) it is important to know how the basis matrix B is constructed.

Internally **MOSEK** employs the linear optimization problem

$$\begin{array}{ll} \text{maximize} & c^T x \\ \text{subject to} & Ax - x^c = 0, \\ & l^x \leq x \leq u^x, \\ & l^c \leq x^c \leq u^c. \end{array} \tag{7.3}$$

where

$$x^c \in \mathbb{R}^m \text{ and } x \in \mathbb{R}^n.$$

The basis matrix is constructed of m columns taken from

$$\begin{bmatrix} A & -I \end{bmatrix}.$$

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

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

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

$$\text{basis}[i] \geq \text{numcon},$$

then the i 'th basis variable is variable

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

and the i 'th column of B is the column

$$A_{\cdot, (\text{basis}[i] - \text{numcon})}.$$

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

An example

Consider the linear optimization problem:

$$\begin{aligned} \text{minimize} \quad & x_0 + x_1 \\ \text{subject to} \quad & x_0 + 2x_1 \leq 2, \\ & x_0 + x_1 \leq 6, \\ & x_0, x_1 \geq 0. \end{aligned} \tag{7.4}$$

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

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

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

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

Please note the ordering of the columns in B .

The program in Listing 7.3 demonstrates the use of *Task.solvewithbasis*.

Listing 7.3: A program showing how to identify the basis.

```
package com.mosek.example;
import mosek.*;

public class solvebasis {
    public static void main(String[] args) {
        // Since the value infinity is never used, we define
        // 'infinity' symbolic purposes only
        double
        infinity = 0;
```

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

int[]   ptrb = {0, 2};
int[]   ptre = {2, 4};

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

double[] aval = {1.0, 1.0,
               2.0, 1.0
               };

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

double[] buc = {2.0,
               6.0
               };

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

int    numvar = 2;
int    numcon = 2;

double[] w1 = {2.0, 6.0};
double[] w2 = {1.0, 0.0};

try (Env env = new Env();
     Task task = new Task(env, 0, 0)) {
    task.inputdata(numcon, numvar,
                  c,
                  0.0,
                  ptrb,
                  ptre,
                  asub,
                  aval,
                  bkc,
                  blc,
                  buc,
                  bkc,
                  blx,
                  bux);
    task.putobjsense(mosek.objsense.maximize);

    System.out.println("optimize");
    try {
        task.optimize();
    }
```



```

} catch (mosek.Warning e) {
    System.out.println("Mosek warning:");
    System.out.println(e.toString());
}

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

//List basis variables corresponding to columns of B
int[] varsub = {0, 1};
for (int i = 0; i < numcon; i++) {
    System.out.println("Basis i:" + i + " Basis:" + basis[i]);
    if (basis[varsub[i]] < numcon) {
        System.out.println("Basis variable no " + i + " is xc" +
            basis[i]);
    } else {
        int index = basis[i] - numcon;
        System.out.println("Basis variable no " + i + " is x" +
            index);
    }
}

// solve Bx = w1
// varsub contains index of non-zeros in b.
// On return b contains the solution x and
// varsub the index of the non-zeros in x.

int[] nz = new int[1];
nz[0] = 2;

task.solvewithbasis(0, nz, varsub, w1);
System.out.println("nz =" + nz[0]);
System.out.println("\nSolution to Bx = w1:\n");

for (int i = 0; i < nz[0]; i++) {
    if (basis[varsub[i]] < numcon) {
        System.out.println("xc" + basis[varsub[i]] + "=" + w1[varsub[i]]);
    } else {
        int index = basis[varsub[i]] - numcon;
        System.out.println("x" + index + " = " + w1[varsub[i]]);
    }
}

// Solve B^Tx = w2
nz[0] = 2;
varsub[0] = 0;
varsub[1] = 1;

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

System.out.println("\nSolution to B^Tx = w2:\n");

for (int i = 0; i < nz[0]; i++) {
    if (basis[varsub[i]] < numcon) {
        System.out.println("xc" + basis[varsub[i]] + " = " + w2[varsub[i]]);
    } else {
        int index = basis[varsub[i]] - numcon;
        System.out.println("x" + index + " = " + w2[varsub[i]]);
    }
}

} catch (mosek.Exception e)
    /* Catch both Error and Warning */

```

```

    {
        System.out.println("An error was encountered");
        System.out.println(e.getMessage());
        throw e;
    }
}
}

```

In the example above the linear system is solved using the optimal basis for (7.4) and the original right-hand side of the problem. Thus the solution to the linear system is the optimal solution to the problem. When running the example program the following output is produced.

```

basis[0] = 1
Basis variable no 0 is xc1.
basis[1] = 2
Basis variable no 1 is x0.

Solution to Bx = b:

x0 = 2.000000e+00
xc1 = -4.000000e+00

Solution to B^Tx = c:

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

```

Please note that the ordering of the basis variables is

$$\begin{bmatrix} x_1^c \\ x_0 \end{bmatrix}$$

and thus the basis is given by:

$$B = \begin{bmatrix} 0 & 1 \\ -1 & 1 \end{bmatrix}$$

It can be verified that

$$\begin{bmatrix} x_1^c \\ x_0 \end{bmatrix} = \begin{bmatrix} -4 \\ 2 \end{bmatrix}$$

is a solution to

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

7.2.2 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

$$\begin{bmatrix} 0 & 1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \quad (7.5)$$

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

```

package com.mosek.example;
import mosek.*;

public class solvelinear {
    static public void put_a(
        mosek.Task task,
        double[] [] aval,
        int[] [] asub,
        int[] ptrb,
        int[] ptre,
        int numvar,
        int[] basis ) {
        // Since the value infinity is never used, we define
        // 'infinity' symbolic purposes only
        double
        infinity = 0;

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

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

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

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

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

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

        //task.makesolutionstatusunknown(mosek.soltype.bas);

        /* Define a basic solution by specifying
           status keys for variables & constraints. */

        for (int i = 0 ; i < numvar ; ++i)
            task.putsolutioni (
                mosek.accmode.var,
                i,
                mosek.soltype.bas,
                skx[i],
                0.0,
                0.0,
                0.0,
                0.0);
    }
}

```

```

    for (int i = 0 ; i < numvar ; ++i)
        task.putsolutioni (
            mosek.accmode.con,
            i,
            mosek.soltype.bas,
            skc[i],
            0.0,
            0.0,
            0.0,
            0.0);

    task.initbasissolve(basis);
}

public static void main (String[] argv) {
    int numcon = 2;
    int numvar = 2;

    double[][] aval = {
        { -1.0 },
        { 1.0, 1.0 }
    };
    int[][] asub = {
        { 1 },
        { 0, 1 }
    };
    int [] ptrb = new int[] {0, 1};
    int [] ptre = new int[] {1, 3};

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

    try (Env env = new Env();
        Task task = new Task(env, 0, 0)) {
        // Directs the log task stream to the user specified
        // method task_msg_obj.streamCB
        task.set_Stream(
            mosek.streamtype.log,
            new mosek.Stream()
            { public void stream(String msg) { System.out.print(msg); } });

        /* Put A matrix and factor A.
           Call this function only once for a given task. */

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

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

```

```

task.solvewithbasis(0, nz_, bsub, b);
int nz = nz_[0];
System.out.println("\nSolution to Bx = b:\n");

/* Print solution and show correspondents
   to original variables in the problem */
for (int i = 0; i < nz; ++i) {
    if (basis[bsub[i]] < numcon)
        System.out.println ("This should never happen");
    else
        System.out.println("x" + (basis[bsub[i]] - numcon) + " = " + b[bsub[i]]);
}

b[0] = 7;
bsub[0] = 0;
nz_[0] = 1;
task.solvewithbasis(0, nz_, bsub, b);
nz = nz_[0];

System.out.println ("\nSolution to Bx = b:\n");
/* Print solution and show correspondents
   to original variables in the problem */
for (int i = 0; i < nz; ++i) {
    if (basis[bsub[i]] < numcon)
        System.out.println("This should never happen");
    else
        System.out.println("x" + (basis[bsub[i]] - numcon) + " = " + b[bsub[i]] );
}
}
}
}
}

```

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 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 (7.5).

7.3 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 7.1](#) we list BLAS functions.

Table 7.1: 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 7.2](#).

Table 7.2: 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 7.4](#) 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.

Listing 7.4: A working example on how to call BLAS and LAPACK routines from MOSEK.

```
package com.mosek.example;

public class blas_lapack {
    static final int n = 3, m = 2, k = 3;

    public static void main (String[] args) {

        double alpha = 2.0, beta = 0.5;
        double[] x = {1., 1., 1.};
        double[] y = {1., 2., 3.};
        double[] z = {1.0, 1.0};

        /*A has m=2 rows and k=3 cols*/
        double[] A = {1., 1., 2., 2., 3., 3.};
        /*B has k=3 rows and n=3 cols*/
        double[] B = {1., 1., 1., 1., 1., 1., 1., 1., 1.};
        double[] C = { 1., 2., 3., 4., 5., 6.};

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

        double[] xy = {0.};

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

            env.dot(n, x, y, xy);
        }
    }
}
```

```

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

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

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

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

/* LAPACK routines*/

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

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

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

} catch (mosek.Exception e) {
    System.out.println("An error/warning was encountered");
    System.out.println(e.toString());
}
}
}

```

7.4 Computing a Sparse Cholesky Factorization

Given a positive semidefinite symmetric (PSD) matrix

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

it is well known there exists a matrix L such that

$$A = LL^T.$$

If the matrix L is lower triangular then it is called a *Cholesky factorization*. Given A is positive definite (nonsingular) then L is also nonsingular. A Cholesky factorization is useful for many reasons:

- A system of linear equations $Ax = b$ can be solved by first solving the lower triangular system $Ly = b$ followed by the upper triangular system $L^T x = y$.
- A quadratic term $x^T Ax$ in a constraint or objective can be replaced with $y^T y$ for $y = L^T x$, potentially leading to a more robust formulation (see [\[And13\]](#)).

Therefore, **MOSEK** provides a function that can compute a Cholesky factorization of a PSD matrix. In addition a function for solving linear systems with a nonsingular lower or upper triangular matrix is available.

In practice A may be very large with n is in the range of millions. However, then A is typically sparse which means that most of the elements in A are zero, and sparsity can be exploited to reduce the cost of computing the Cholesky factorization. The computational savings depend on the positions of zeros in A . For example, below a matrix A is given together with a Cholesky factor up to 5 digits of accuracy:

$$A = \begin{bmatrix} 4 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}, \quad L = \begin{bmatrix} 2.0000 & 0 & 0 & 0 \\ 0.5000 & 0.8660 & 0 & 0 \\ 0.5000 & -0.2887 & 0.8165 & 0 \\ 0.5000 & -0.2887 & -0.4082 & 0.7071 \end{bmatrix}. \quad (7.6)$$

However, if we symmetrically permute the rows and columns of A using a permutation matrix P

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}, \quad A' = PAP^T = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 4 \end{bmatrix},$$

then the Cholesky factorization of $A' = L'L'^T$ is

$$L' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

which is sparser than L .

Computing a permutation matrix that leads to the sparsest Cholesky factorization or the minimal amount of work is NP-hard. Good permutations can be chosen by using heuristics, such as the minimum degree heuristic and variants. The function `Env.computesparscholesky` provided by **MOSEK** for computing a Cholesky factorization has a built in permutation aka. reordering heuristic. The following code illustrates the use of `Env.computesparscholesky` and `Env.sparsetriangularsolvedense`.

Listing 7.5: How to use the sparse Cholesky factorization routine available in **MOSEK**.

```

env.computesparscholesky(0,          //Disable multithreading
                        1,          //Apply reordering heuristic
                        1.0e-14,    //Singularity tolerance
                        anzc, aptrc, asubc, avalc,
                        perm, diag,
                        lnzc, lptrc, lensubnval, lsubc, lvalc);

printspars(n, perm[0], diag[0], lnzc[0], lptrc[0], lensubnval[0], lsubc[0], lvalc[0]);

/* Permuted b is stored as x. */
double[] x = new double[n];
for (int i = 0; i < n; i++) x[i] = b[perm[0][i]];

/*Compute inv(L)*x.*/
env.sparsetriangularsolvedense(mosek.transpose.no, lnzc[0], lptrc[0], lsubc[0],
→lvalc[0], x);
/*Compute inv(L^T)*x.*/
env.sparsetriangularsolvedense(mosek.transpose.yes, lnzc[0], lptrc[0], lsubc[0],
→lvalc[0], x);

System.out.print("\nSolution A x = b, x = [ ");
for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++) if (perm[0][j] == i) System.out.print(x[j] + " ");
System.out.println("\n");

```

We can set up the data to recreate the matrix A from (7.6):

```

//Observe that anzc, aptrc, asubc and avalc only specify the lower triangular part.
int n = 4;
int[] anzc = {4, 1, 1, 1};
int[] asubc = {0, 1, 2, 3, 1, 2, 3};
long[] aptrc = {0, 4, 5, 6};
double[] avalc = {4.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0};
double[] b = {13.0, 3.0, 4.0, 5.0};

```

and we obtain the following output:

```

Example with positive definite A.
P = [ 3 2 0 1 ]
diag(D) = [ 0.00 0.00 0.00 0.00 ]
L=
1.00 0.00 0.00 0.00
0.00 1.00 0.00 0.00
1.00 1.00 1.41 0.00
0.00 0.00 0.71 0.71

Solution A x = b, x = [ 1.00 2.00 3.00 4.00 ]

```


The output indicates that with the permutation matrix

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

there is a Cholesky factorization $PAP^T = LL^T$, where

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

The remaining part of the code solves the linear system $Ax = b$ for $b = [13, 3, 4, 5]^T$. The solution is reported to be $x = [1, 2, 3, 4]^T$, which is correct.

The second example shows what happens when we compute a sparse Cholesky factorization of a singular matrix. In this example A is a rank 1 matrix

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}^T \quad (7.7)$$

```
int n           = 3;
int[] anzc      = {3, 2, 1};
int[] asubc     = {0, 1, 2, 1, 2, 2};
long[] aptrc    = {0, 3, 5, };
double[] avalc  = {1.0, 1.0, 1.0, 1.0, 1.0, 1.0};
```

Now we get the output

```
P = [ 0 2 1 ]
diag(D) = [ 0.00e+00 1.00e-14 1.00e-14 ]
L=
1.00e+00 0.00e+00 0.00e+00
1.00e+00 1.00e-07 0.00e+00
1.00e+00 0.00e+00 1.00e-07
```

which indicates the decomposition

$$PAP^T = LL^T - D$$

where

$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \quad L = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 10^{-7} & 0 \\ 1 & 0 & 10^{-7} \end{bmatrix}, \quad D = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 10^{-14} & 0 \\ 0 & 0 & 10^{-14} \end{bmatrix}.$$

Since A is only positive semidefinite, but not of full rank, some of diagonal elements of A are boosted to make it truly positive definite. The amount of boosting is passed as an argument to `Env.computesparsecholesky`, in this case 10^{-14} . Note that

$$PAP^T = LL^T - D$$

where D is a small matrix so the computed Cholesky factorization is exact of slightly perturbed A . In general this is the best we can hope for in finite precision and when A is singular or close to being singular.

We will end this section by a word of caution. Computing a Cholesky factorization of a matrix that is not of full rank and that is not sufficiently well conditioned may lead to incorrect results i.e. a matrix that is indefinite may declared positive semidefinite and vice versa.

7.5 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

$$\sum_{j=2}^n x_j^2 - x_1^2 \leq 0.0 \quad (7.8)$$

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 computational expensive.
- the left hand side of (7.8) is nonconvex so the formulation is strictly 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 specify 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:

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

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.

7.5.1 Quadratic Constraint Reformulation

Let us assume we want to convert the following quadratic constraint

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

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

$$\frac{1}{2}x^T Qx + \sum_{j=0}^{n-1} a_j^T x_j \leq u. \quad (7.9)$$

Introducing an additional variable w such that

$$w = u - \sum_{j=0}^{n-1} a_j^T x_j \quad (7.10)$$

we obtain the equivalent form

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

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

$$Q = FF^T \quad (7.11)$$

and therefore we can write

$$\begin{aligned} \|Fx\|^2 &\leq 2w, \\ u - \sum_{j=0}^{n-1} a_j^T x_j &= w. \end{aligned}$$

Introducing an additional variable $z = 1$, and setting $y = Fx$ we obtain the conic formulation

$$\begin{aligned} (w, z, y) &\in \mathcal{Q}_r, \\ z &= 1, \\ y &= Fx \\ w &= u - \sum_{j=0}^{n-1} a_j^T x_j. \end{aligned} \quad (7.12)$$

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

1. a rotated quadratic cone of dimension $t + 2$,
2. two additional variables for the cone roots,
3. t additional variables to map the remaining part of the cone,
4. t linear constraints.

7.5.2 Some Examples

We report in this section few examples of reformulation of a QCQO 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

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

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

[variables 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 $x^* = (1, 0.5, -1)$. The conversion will introduce first a variable x_3 in the objective function such that $x_3 \geq 1/2x^T Qx$ and then convert the latter directly in conic form. The converted problem follows:

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

The model generated by *Task.toconic* is

```
[comment]
Written by MOSEK version 8.0.0.8
Date 25-05-15
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 (7.11), while (7.10) corresponds to *c0003*. The cone roots are *x0005* and *x0004*.

7.6 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.

7.6.1 Synchronous Remote Optimization

In synchronous mode 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 7.6 we show how to modify tutorial in Section 5.1 so that the computation is off loaded to a remote machine.

Listing 7.6: Using the OptServer in synchronous mode.

```

package com.mosek.example;
import mosek.*;

public class opt_server_sync {
    public static void main (String[] args) {
        if (args.length == 0) {
            System.out.println ("Missing argument, syntax is:");
            System.out.println ("  opt_server_sync inputfile host port numpolls");
        } else {

            String inputfile = args[0];
            String host      = args[1];

```

```

String port      = args[2];

rescode trm[] = new rescode[1];

try (Env env = new Env();
     Task task = new Task(env, 0, 0)) {
    task.set_Stream (mosek.streamtype.log,
        new mosek.Stream() {
            public void stream(String msg) { System.out.print(msg); }
        });

    task.readdata (inputfile);

    task.optimizermt (host, port, trm);

    task.solutionssummary (mosek.streamtype.log);
}
}
}
}

```

7.6.2 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 7.7 the code in Listing 7.6 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 7.7: Using the OptServer in synchronous mode.

```

package com.mosek.example;
import mosek.*;

public class opt_server_async {
    public static void main (String[] args) {
        if (args.length == 0) {
            System.out.println ("Missing argument, syntax is:");
            System.out.println ("  opt_server_async inputfile host port numpolls");
        } else {

            String inputfile = args[0];
            String host      = args[1];
            String port      = args[2];
            int numpolls     = Integer.parseInt(args[3]);

            try (Env env = new Env()) {
                String token;

                try (Task task = new Task(env, 0, 0)) {
                    task.readdata (inputfile);
                    token = task.asyncoptimize (host, port);
                }
            }
        }
    }
}

```

```

}

System.out.printf("Task token = %s\n", token);

try(Task task = new Task(env, 0, 0)) {
    System.out.println("Reading input file...");

    task.readdata (inputfile);

    System.out.println("Setting log stream...");

    task.set_Stream (mosek.streamtype.log,
        new mosek.Stream() {
            public void stream(String msg) { System.out.print(msg); }
        });

    long start = System.currentTimeMillis();

    System.out.println("Starting polling loop...");

    int i = 0;

    while ( true ) {

        Thread.sleep(100);

        System.out.printf("poll %d...\n", i);

        rescode trm[] = new rescode[1];
        rescode resp[] = new rescode[1];

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

        System.out.println("polling done");

        if (respavailable) {
            System.out.println("solution available!");
            task.asyncgetresult(host,
                                port,
                                token,
                                resp,
                                trm);

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

        i++;

        if (i == numpolls) {
            System.out.println("max num polls reached, stopping host.");
            task.asyncstop (host, port, token);
            break;
        }

    }

} catch (java.lang.Exception e) {
    System.out.println("Something unexpected happend...");
}

```

```
}  
  }  
  }  
}
```


GUIDELINES

8.1 Deployment

When redistributing a Java application using the **MOSEK** Optimizer API for Java 8.1.0.15(beta), the following libraries must be included:

64-bit Linux	64-bit Windows	32-bit Windows	64-bit Mac OS
libmosek64.so.8.1	mosek64_8_1.dll	mosek32_8_1.dll	libmosek64.8.1.dylib
libomp5.lib	libomp5md.dll	libomp5md.dll	libomp5.dylib
libcilkrts.so.5	cilkrts20.dll	cilkrts20.dll	libcilkrts.5.dylib
libmosekjava8_1.so	mosekjava8_1.dll	mosekjava8_1.dll	libmosekjava8_1.dylib
libmosekxx8_1.so	mosekxx8_1.dll	mosekxx8_1.dll	libmosekxx8_1.dylib
libmosekscopt8_1.so	mosekscopt8_1.dll	mosekscopt8_1.dll	libmosekscopt8_1.dylib

By default the Java interface will look for the binaries in the same directory as the `.jar` file, so they should be placed in the same directory when redistributing.

8.2 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.
- `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`.

8.2.1 API overhead

The Java interface is a thin wrapper around a native **MOSEK** library. The layer between the Java 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 library `mosekjava`. Each method converts the call parameter data structures (i.e. creates a complete copy of the data), calls a native **MOSEK** function and converts the returned values back into Java structures.

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

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

- 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 dispose task and environment when not in use anymore This can best be done by using a construction as

```

Env e = new Env();
try
{
    // use env here
} finally
{
    e.dispose();
}

```

- Avoid input loops

Whenever possible input 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.putqobjj`.

8.3 The license system

MOSEK is a commercial product that **always** needs a valid license to work. **MOSEK** uses a third party license manager to implement license checking. The number of license tokens provided determines the number of optimizations that can be run simultaneously.

By default a license token remains checked out from the first optimization until the end of the **MOSEK** session, i.e.

- a license token is checked out when `Task.optimize` is first called, and
- it is returned when the **MOSEK** environment is deleted.

Calling `Task.optimize` from different threads using the same **MOSEK** environment only consumes one license token.

Starting the optimization when no license tokens are available will result in an error.

Default behaviour of the license system can be changed in several ways:

- Setting the parameter `iparam.cache_license` to `onoffkey.off` will force **MOSEK** to return the license token immediately after the optimization completed.
- Setting the license wait flag with the parameter `iparam.license_wait` will force **MOSEK** to wait until a license token becomes available instead of returning with an error. The wait time between checks can be set with `Env.putlicensewait`.
- Additional license checkouts and checkins can be performed with the functions `Env.checkinlicense` and `Env.checkoutlicense`.
- Usually the license system is stopped automatically when the **MOSEK** library is unloaded. However, when the user explicitly unloads the library (using e.g. `FreeLibrary`), the license system must be stopped before the library is unloaded. This can be done by calling the function `Env.licensecleanup` as the last function call to **MOSEK**.

CASE STUDIES

In this section we present some case studies in which the Optimizer API for Java is used to solve real-life applications. These examples involve some more advanced modelling 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
<i>Portfolio Optimization</i>	CQO	NO	Markowitz, Slippage, Market Impact

9.1 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

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

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 .

9.1.1 A Basic Portfolio Optimization Model

The classical Markowitz portfolio optimization problem considers investing in n stocks or assets held over a period of time. Let x_j denote the amount invested in asset j , and assume a stochastic model where the return of the assets is a random variable r with known mean

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

and covariance

$$\Sigma = \mathbf{E}(r - \mu)(r - \mu)^T.$$

The return of the investment is also a random variable $y = r^T x$ with mean (or expected return)

$$\mathbf{E}y = \mu^T x$$

and variance (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

$$\begin{aligned} & \text{maximize} && \mu^T x \\ & \text{subject to} && e^T x = w + e^T x^0, \\ & && x^T \Sigma x \leq \gamma^2, \\ & && x \geq 0. \end{aligned} \tag{9.1}$$

The variables x denote the investment i.e. x_j is the amount invested in asset j and x_j^0 is the initial holding of asset j . Finally, w is the initial amount of cash available.

A popular choice is $x^0 = 0$ and $w = 1$ because then x_j may be interpreted as the relative amount of the total portfolio that is invested in asset j .

Since e is the vector of all ones then

$$e^T x = \sum_{j=1}^n x_j$$

is the total investment. Clearly, the total amount invested must be equal to the initial wealth, which is

$$w + e^T x^0.$$

This leads to the first constraint

$$e^T x = w + e^T x^0.$$

The second constraint

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

ensures that the variance, 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

$$\Sigma = GG^T. \tag{9.2}$$

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 9.1.3.

For a given G we have that

$$\begin{aligned} x^T \Sigma x &= x^T GG^T x \\ &= \|G^T x\|^2. \end{aligned}$$

Hence, we may write the risk constraint as

$$\gamma \geq \|G^T x\|$$

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 (9.1) can be written as

$$\begin{aligned} & \text{maximize} && \mu^T x \\ & \text{subject to} && e^T x = w + e^T x^0, \\ & && [\gamma; G^T x] \in \mathcal{Q}^{n+1}, \\ & && x \geq 0, \end{aligned} \tag{9.3}$$

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 (9.1) 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 (9.1) 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

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

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.

Implementing the Portfolio Model

Model (9.3) can not be implemented as stated using the **MOSEK** optimizer API because the API requires the problem to be on the form

$$\begin{aligned} & \text{maximize} && c^T \hat{x} \\ & \text{subject to} && l^c \leq A\hat{x} \leq u^c, \\ & && l^x \leq \hat{x} \leq u^x, \\ & && \hat{x} \in \mathcal{K}. \end{aligned} \quad (9.4)$$

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

The first step in bringing (9.3) to the form (9.4) is the reformulation

$$\begin{aligned} & \text{maximize} && \mu^T x \\ & \text{subject to} && e^T x = w + e^T x^0, \\ & && G^T x - t = 0 \\ & && [s; t] \in \mathcal{Q}^{n+1}, \\ & && x \geq 0, \\ & && s \geq 0. \end{aligned} \quad (9.5)$$

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

$$\hat{x} = [x; s; t] = \begin{bmatrix} x \\ s \\ t \end{bmatrix}. \quad (9.6)$$

Hence, the API variable \hat{x} is concatenation of model variables x , s and t . In Table 9.1 the details of the concatenation are specified.

Table 9.1: 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 (9.6) the data should be defined as follows

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

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

$$\begin{aligned}
 & \text{for} && j = 1 : n \\
 & && \hat{x}_j = x_j \\
 & && A_{1,j} = 1.0 \\
 & && A_{2:(n+1),j} = G_{j,1:n}^T \\
 \\
 & && \hat{x}_{n+1} = s \\
 \\
 & \text{for} && j = 1 : n \\
 & && \hat{x}_{n+1+j} = t_j \\
 & && A_{n+1+j,n+1+j} = -1.0
 \end{aligned}$$

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 Java programming language employs 0 as index origin and that should be kept in mind when reading the example code.

Listing 9.1: Code implementing model (9.3).

```

package com.mosek.example;

public class case_portfolio_1 {
    static final int n = 3;

    public static void main (String[] args) {
        // Since the value infinity is never used, we define
        // 'infinity' symbolic purposes only
        double infinity = 0;
        double gamma = 0.05;
        double[] mu = {0.1073, 0.0737, 0.0627};
        double[][] GT = {
            {0.1667, 0.0232, 0.0013},
            {0.0000, 0.1033, -0.0022},
            {0.0000, 0.0000, 0.0338}
        };
        double[] x0 = {0.0, 0.0, 0.0};
        double w = 1.0;

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

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

        try ( mosek.Env env = new mosek.Env ();
              mosek.Task task = new mosek.Task (env, 0, 0) ) {
            // Directs the log task stream to the user specified
            // method task_msg_obj.stream
            task.set_Stream(
                mosek.streamtype.log,
                new mosek.Stream()
            );
            { public void stream(String msg) { System.out.print(msg); } });

            //Constraints.
            task.appendcons(numcon);
            for ( int i = 1; i <= n; ++i ) {
                w += x0[i - 1];
                task.putconbound(i, mosek.boundkey.fx, 0., 0.);
            }
        }
    }
}

```

```

    task.putconname(i, "GT[" + i + "]");
}
task.putconbound(0, mosek.boundkey.fx, w, w);
task.putconname(0, "budget");

//Variables.
task.appendvars(numvar);

int[] xindx = {offsetx + 0, offsetx + 1, offsetx + 2};
task.putclist(xindx, mu);

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

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

    task.putvarbound(offsetx + i, mosek.boundkey.lo, 0., 0.);

    task.putvarname(offsetx + i, "x[" + (i + 1) + "]");
    task.putvarname(offsett + i, "t[" + (i + 1) + "]");
    task.putvarbound(offsett + i, mosek.boundkey.fr, 0., 0.);
}
task.putvarbound(offsets, mosek.boundkey.fx, gamma, gamma);
task.putvarname(offsets, "s");

double[] e = {1.0, 1.0, 1.0};
task.putarow(0, xindx, e);

//Cones.
int[] csub = {offsets, offsett + 0, offsett + 1, offsett + 2};
task.appendcone( mosek.conetype.quad,
                0.0, /* For future use only, can be set to 0.0 */
                csub);
task.putconename(0, "stddev");

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

//task.solutionsummary(mosek.streamtype.log);

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

/* Solve the problem */
try {

    task.optimize();

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

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

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

    System.out.printf("\nExpected return %e for gamma %e\n", expret, xx[offsets]);

} catch (mosek.Warning mw) {
    System.out.println (" Mosek warning:");
    System.out.println (mw.toString ());
}
} catch ( mosek.Exception e) {

```

```

        System.out.println ("An error/warning was encountered");
        System.out.println (e.toString ());
        throw e;
    }
}
}

```

The above code produce the result

Listing 9.2: 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.

In the lines

```

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

```

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

```

int[] xindx = {offsetx + 0, offsetx + 1, offsetx + 2};
task.putclist(xindx, mu);

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

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

    task.putvarbound(offsetx + i, mosek.boundkey.lo, 0., 0.);

    task.putvarname(offsetx + i, "x[" + (i + 1) + "]");
    task.putvarname(offsett + i, "t[" + (i + 1) + "]");
    task.putvarbound(offsett + i, mosek.boundkey.fr, 0., 0.);
}
task.putvarbound(offsets, mosek.boundkey.fx, gamma, gamma);
task.putvarname(offsets, "s");

double[] e = {1.0, 1.0, 1.0};
task.putarow(0, xindx, e);

```

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

```

int[] xindx = {offsetx + 0, offsetx + 1, offsetx + 2};
task.putclist(xindx, mu);

```

inputs the objective coefficients for the **x** variables. Moreover, the code

```

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

```

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 9.3: Problem (9.5) 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]'
[/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.

9.1.2 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

$$\begin{aligned} & \text{maximize} && \mu^T x - \alpha s \\ & \text{subject to} && e^T x = w + e^T x^0, \\ & && [s; G^T x] \in \mathcal{Q}^{n+1}, \\ & && x \geq 0. \end{aligned} \tag{9.7}$$

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

Using the example data from Section 9.1.1, the optimal values of return and risk for several α s are listed below:

Listing 9.4: Results obtained solving problem (9.7) 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

The example code in Listing 9.5 demonstrates how to compute the efficient portfolios for several values of α .

Listing 9.5: Code implementing model (9.7).

```
package com.mosek.example;

import mosek.*;

public class case_portfolio_2 {
    static final int n = 3;

    public static void main (String[] args) {
        // Since the value infinity is never used, we define
        // 'infinity' symbolic purposes only
        double infinity = 0;
        double gamma = 0.05;
        double[] mu = {0.1073, 0.0737, 0.0627};
        double[][] GT = {
            {0.1667, 0.0232, 0.0013},
            {0.0000, 0.1033, -0.0022},
            {0.0000, 0.0000, 0.0338}
        };
        double[] x0 = {0.0, 0.0, 0.0};
        double w = 1.0;
        double[] alphas = {0.0, 0.25, 0.5, 0.75, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5};
        int numalphas = 12;

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

```

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

try ( Env env = new mosek.Env ();
      Task task = new mosek.Task (env, 0, 0) ) {

    // Directs the log task stream to the user specified
    // method task_msg_obj.stream
    task.set_Stream(
        mosek.streamtype.log,
        new mosek.Stream()
    { public void stream(String msg) { System.out.print(msg); } });

    //Constraints.
    task.appendcons(numcon);
    for ( int i = 1; i <= n; ++i) {
        w += x0[i - 1];
        task.putconbound(i, mosek.boundkey.fx, 0., 0.);
        task.putconname(i, "GT[" + i + "]");
    }
    task.putconbound(0, mosek.boundkey.fx, w, w);
    task.putconname(0, "budget");

    //Variables.
    task.appendvars(numvar);

    int[] xindx = {offsetx + 0, offsetx + 1, offsetx + 2};
    task.putclist(xindx, mu);

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

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

        task.putvarbound(offsetx + i, mosek.boundkey.lo, 0., 0.);

        task.putvarname(offsetx + i, "x[" + (i + 1) + "]");
        task.putvarname(offsett + i, "t[" + (i + 1) + "]");
        task.putvarbound(offsett + i, mosek.boundkey.fr, 0., 0.);
    }
    task.putvarbound(offsets, mosek.boundkey.fr, gamma, gamma);
    task.putvarname(offsets, "s");

    double[] e = {1.0, 1.0, 1.0};
    task.putarow(0, xindx, e);

    //Cones.
    int[] csub = {offsets, offsett + 0, offsett + 1, offsett + 2};
    task.appendcone( mosek.conetype.quad,
        0.0, /* For future use only, can be set to 0.0 */
        csub);
    task.putconename(0, "stddev");

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

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

```

```

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

    System.out.printf("%-12s  %-12s  %-12s", "alpha", "exp ret", "std dev");
    for (int k = 0; k < numalphas; ++k) {
        task.putcj(offsets, -alphas[k]);

        task.optimize();

        task.solutionsummary(mosek.streamtype.log);

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

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

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

        System.out.printf("%-12.3e  %-12.3e  %-12.3e\n", alphas[k], expret, xx[offsets]);
    }
    System.out.println("");
} catch (mosek.Warning mw) {
    System.out.println (" Mosek warning:");
    System.out.println (mw.toString ());
}
} catch ( mosek.Exception e) {
    System.out.println ("An error/warning was encountered");
    System.out.println (e.toString ());
    throw e;
}
}
}

```

9.1.3 Improving the Computational Efficiency

In practice it is often important to solve the portfolio problem in a short amount of time; this section it 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 of nonzeros used to represent the problem. Indeed it is often better to focus at the number of nonzeros in G (see (9.2)) 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 to be a small number, say less than 100, independent of n .

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.

9.1.4 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

$$\begin{aligned} & \text{maximize} && \mu^T x \\ & \text{subject to} && e^T x + \sum_{j=1}^n C_j(x_j - x_j^0) = w + e^T x^0, \\ & && x^T \Sigma x \leq \gamma^2, \\ & && x \geq 0, \end{aligned} \tag{9.8}$$

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 asset 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

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

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

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

Unfortunately this set of constraints is nonconvex due to the constraint

$$z_j = |x_j - x_j^0| \tag{9.9}$$

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

$$z_j \geq |x_j - x_j^0| \quad (9.10)$$

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

$$z_j > |x_j - x_j^0| \quad (9.11)$$

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 (9.11) 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 (9.9) and (9.10) are equivalent.

Formula (9.10) is replaced by constraints

$$\begin{aligned} z_j &\geq x_j - x_j^0, \\ z_j &\geq -(x_j - x_j^0). \end{aligned} \quad (9.12)$$

Now we have

$$\begin{aligned} &\text{maximize} && \mu^T x \\ &\text{subject to} && e^T x + m^T c = w + e^T x^0, \\ & && z_j \geq x_j - x_j^0, \quad j = 1, \dots, n, \\ & && z_j \geq x_j^0 - x_j, \quad j = 1, \dots, n, \\ & && [\gamma; G^T x] \in \mathcal{Q}^{n+1}, \\ & && [v_j; c_j; z_j] \in \mathcal{Q}_r^3, \quad j = 1, \dots, n, \\ & && [z_j; 1/8; v_j] \in \mathcal{Q}_r^3, \quad j = 1, \dots, n, \\ & && x \geq 0. \end{aligned} \quad (9.13)$$

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

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

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 (9.13) is needed,

$$\begin{aligned}
& \text{maximize} && \mu^T x \\
& \text{subject to} && e^T x + m^T c = w + e^T x^0, \\
& && G^T x - t = 0, \\
& && z_j - x_j \geq -x_j^0, & j = 1, \dots, n, \\
& && z_j + x_j \geq x_j^0, & j = 1, \dots, n, \\
& && [v_j; c_j; z_j] - f_{j,1:3} = 0, & j = 1, \dots, n, \\
& && [z_j; 0; v_j] - g_{j,1:3} = [0; -1/8; 0], & j = 1, \dots, n, \\
& && [s; t] \in \mathcal{Q}^{n+1}, \\
& && f_{j,1:3}^T \in \mathcal{Q}_r^3, & j = 1, \dots, n, \\
& && g_{j,1:3}^T \in \mathcal{Q}_r^3, & j = 1, \dots, n, \\
& && x \geq 0, \\
& && s = \gamma,
\end{aligned} \tag{9.14}$$

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 (9.14) is not the most compact possible. However, the **MOSEK** presolve will automatically make it more compact and since it is easier to implement (9.14) than a more compact form then the form (9.14) is preferred.

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

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

will be used. Note $f^T(\cdot)$ means the rows of f are transposed and stacked on top of each other to form a long column vector. Table 9.2 shows the mapping between the \hat{x} and the model variables.

Table 9.2: 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	$4n+2$
$f(\cdot)^T$	$3n$	$7n+2$
$g(\cdot)^T$	$3n$	$10n+2$

The next step is to consider how the columns of A is defined. Reusing the idea in Section 9.1.1 then the

following pseudo code describes the setup of A .

```

for       $j = 1 : n$ 
     $\hat{x}_j = x_j$ 
     $A_{1,j} = 1.0$ 
     $A_{2:n+1,j} = G_{j,1:n}^T$ 
     $A_{n+1+j,j} = -1.0$ 
     $A_{2n+1+j,j} = 1.0$ 

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

for       $j = 1 : n$ 
     $\hat{x}_{n+1+j} = t_j$ 
     $A_{1+j,n+1+j} = -1.0$ 

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

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

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

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

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

```

The example code in [Listing 9.6](#) demonstrates how to implement the model (9.14).

Listing 9.6: Code implementing model (9.14).

```

package com.mosek.example;

public class case_portfolio_3 {
    static final int n = 3;

    public static void main (String[] args) {
        // Since the value infinity is never used, we define
        // 'infinity' symbolic purposes only
    }
}

```

```

double infinity = 0;
double gamma = 0.05;
double[] mu = {0.1073, 0.0737, 0.0627};
double[][] GT = {
    {0.1667, 0.0232, 0.0013},
    {0.0000, 0.1033, -0.0022},
    {0.0000, 0.0000, 0.0338}
};
double[] x0 = {0.0, 0.0, 0.0};
double w = 1.0;
double[] m = {0.01, 0.01, 0.01};

int offsetx = 0;
int offsets = offsetx + n;
int offsett = offsets + 1;
int offsetc = offsett + n;
int offsetv = offsetc + n;
int offsetz = offsetv + n;
int offsetf = offsetz + n;
int offsetg = offsetf + 3 * n;

int numvar = offsetg + 3 * n;

int offset_con_budget = 0;
int offset_con_gx_t = offset_con_budget + 1;
int offset_con_abs1 = offset_con_gx_t + n;
int offset_con_abs2 = offset_con_abs1 + n;
int offset_con_f = offset_con_abs2 + n;
int offset_con_g = offset_con_f + 3 * n;

int numcon = 1 + 3 * n + 2 * 3 * n;

try (mosek.Env env = new mosek.Env();
    mosek.Task task = new mosek.Task(env, 0, 0)) {

    // Directs the log task stream to the user specified
    // method task_msg_obj.stream
    task.set_Stream(
        mosek.streamtype.log,
        new mosek.Stream()
    {
        public void stream(String msg) { System.out.print(msg); }
    });

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

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

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

        for (int j = 0; j < 3; ++j) {
            task.putconbound(offset_con_f + 3 * i + j, mosek.boundkey.fx, 0., 0.);
            task.putconname(offset_con_f + 3 * i + j, "f[" + (i + 1) + "], " + (j + 1) + "]");

            task.putconbound(offset_con_g + 3 * i + j, mosek.boundkey.fx, 0., 0.);
            task.putconname(offset_con_g + 3 * i + j, "g[" + (i + 1) + "], " + (j + 1) + "]");
        }
    }
}

```

```

    task.putconbound(offset_con_g + 3 * i + 1, mosek.boundkey.fx, -1. / 8, -1. / 8.);
}
// e x = w + e x0
task.putconbound(offset_con_budget, mosek.boundkey.fx, w, w);
task.putconname(offset_con_budget, "budget");

//Variables.
task.appendvars(numvar);

//the objective function coefficients
int[] xindx = {offsetx + 0, offsetx + 1, offsetx + 2};
task.putclist(xindx, mu);

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

//set up variable bounds and names
for ( int i = 0; i < n; ++i) {
    task.putvarbound(offsetx + i, mosek.boundkey.lo, 0., infinity);
    task.putvarbound(offsett + i, mosek.boundkey.fr, infinity, infinity);
    task.putvarbound(offsetc + i, mosek.boundkey.fr, infinity, infinity);
    task.putvarbound(offsetz + i, mosek.boundkey.fr, infinity, infinity);
    task.putvarbound(offsetv + i, mosek.boundkey.fr, infinity, infinity);
    for (int j = 0; j < 3; ++j) {
        task.putvarbound(offsetf + j + i * 3, mosek.boundkey.fr, infinity, infinity);
        task.putvarbound(offsetg + j + i * 3, mosek.boundkey.fr, infinity, infinity);
    }
    task.putvarname(offsetx + i, "x[" + (i + 1) + "]");
    task.putvarname(offsett + i, "t[" + (i + 1) + "]");
    task.putvarname(offsetc + i, "c[" + (i + 1) + "]");
    task.putvarname(offsetz + i, "z[" + (i + 1) + "]");
    task.putvarname(offsetv + i, "v[" + (i + 1) + "]");
    for (int j = 0; j < 3; ++j) {
        task.putvarname(offsetf + j + i * 3, "f[" + (i + 1) + "," + (j + 1) + "]");
        task.putvarname(offsetg + j + i * 3, "g[" + (i + 1) + "," + (j + 1) + "]");
    }

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

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

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

    // z_j - x_j >= -x0_j
    int[] indxf1 = {offsetz + i, offsetx + i};
    task.putarow(offset_con_abs1 + i, indxf1, one_m_one);
    // z_j + x_j >= +x0_j
    int[] indxf2 = {offsetz + i, offsetx + i};
    task.putarow(offset_con_abs2 + i, indxf2, one_one);

    int[] indxf1 = { offsetv + i, offsetf + i * 3};
    task.putarow(offset_con_f + 3 * i, indxf1, one_m_one);
    int[] indxf2 = {offsetc + i, offsetf + i * 3 + 1};
    task.putarow(offset_con_f + 1 + 3 * i, indxf2, one_m_one);
    int[] indxf3 = {offsetz + i, offsetf + i * 3 + 2};
    task.putarow(offset_con_f + 2 + 3 * i, indxf3, one_m_one);

    int[] indxf1 = {offsetz + i, offsetg + i * 3};
    task.putarow(offset_con_g + 3 * i, indxf1, one_m_one);

    task.putaij(offset_con_g + 3 * i + 1, offsetg + i * 3 + 1, -1.);

```

```

    int[] indyg3 = {offsetv + i, offsetg + i * 3 + 2};
    task.putarow(offset_con_g + 3 * i + 2, indyg3, one_m_one);
}
task.putvarbound(offsets, mosek.boundkey.fx, gamma, gamma);
task.putvarname(offsets, "s");

//Cones.
int conecount = 0;

int[] csub = {offsets, offsett + 0, offsett + 1, offsett + 2};
task.appendcone(mosek.conetype.quad, 0.0, csub);
task.putconename(conecount, "stddev");
++conecount;

for (int j = 0; j < n; ++j, ++conecount) {
    int[] coneindx = {offsetf + j * 3, offsetf + j * 3 + 1, offsetf + j * 3 + 2};
    task.appendcone(mosek.conetype.rquad, 0.0, coneindx);
    task.putconename(conecount, "f[" + (j + 1) + "]");
}

for (int j = 0; j < n; ++j, ++conecount) {
    int[] coneindx = {offsetg + j * 3, offsetg + j * 3 + 1, offsetg + j * 3 + 2};
    task.appendcone(mosek.conetype.rquad, 0.0, coneindx);
    task.putconename(conecount, "g[" + (j + 1) + "]");
}
/* A maximization problem */
task.putobjsense(mosek.objsense.maximize);

/* Solve the problem */
try {
    //Turn all log output off.
    //task.putintparam(mosek.iparam.log, 0);

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

    task.optimize();

    task.solutionsummary(mosek.streamtype.log);

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

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

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

    System.out.printf("Expected return %e for gamma %e\n\n", expret, xx[offsets]);
} catch (mosek.Warning mw) {
    System.out.println(" Mosek warning:");
    System.out.println(mw.toString());
}
} catch (mosek.Exception e) {
    System.out.println("An error/warning was encountered");
    System.out.println(e.toString());
    throw e;
}
}
}

```

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 9.7: OPF file for problem (9.14).

```

[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]
  '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]

```

The file verifies that the correct problem has been setup.

ERRORS AND WARNINGS

Interaction between **MOSEK** and the user is not always successful 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.

10.1 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 11) 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.

10.2 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 2)

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

10.2.1 Fatal Error Handling

If **MOSEK** encounter a fatal error caused by either an internal bug or a user error, an *exception* is raised. Therefore it is good practice to setup **try-catch** block in order to handle the exceptions.

10.2.2 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.

10.2.3 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 [5.2](#) for instructions on turning log output on. You should also always catch and handle any exceptions thrown by **MOSEK**.

Please refer to Section [11](#) 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.optimize();
```

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

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 11.1 contains information about the log streams provided by **MOSEK**.
- File I/O is discussed in Section 11.2.
- How to tune the logging verbosity is the topic of Section 11.3.

11.1 Stream I/O

MOSEK execution produces a certain amount of logging at environment and task level. This means that the logging from each environment 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.

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.

The *Stream* class is used to receive text strings emitted to **MOSEK**'s output streams. Extending *Stream* is the way to customize the solver output. For example

```
task.set_Stream(mosek.streamtype.log,
               new mosek.Stream()
               {
                   public void stream(String msg)
                   {
                       System.out.print(msg);
                   }
               });
```

When a *Stream* object is attached to a *Task* stream, any text that is printed to that stream will be passed to the `streamCB` method.

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

The stream can be detached by calling

```
task.set_Stream(mosek.streamtype.log,  
                (mosek.Stream)null);
```

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.

11.2 File I/O

MOSEK supports a range of problem and solution formats listed in Section 19. 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`.

11.3 Verbosity

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

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

Each parameter 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`.

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.

PROBLEM FORMULATION AND SOLUTIONS

In this chapter we will discuss the following issues:

- The formal, mathematical formulations of the problem types that **MOSEK** can solve and their duals.
- The solution information produced by **MOSEK**.
- The infeasibility certificate produced by **MOSEK** if the problem is infeasible.

12.1 Linear Optimization

A linear optimization problem can be written as

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

where

- m is the number of constraints.
- n is the number of decision variables.
- $x \in \mathbb{R}^n$ is a vector of decision variables.
- $c \in \mathbb{R}^n$ is the linear part of the objective function.
- $A \in \mathbb{R}^{m \times n}$ 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 (12.1). If (12.1) has at least one primal feasible solution, then (12.1) is said to be (primal) feasible.

In case (12.1) does not have a feasible solution, the problem is said to be *(primal) infeasible*.

12.1.1 Duality for Linear Optimization

Corresponding to the primal problem (12.1), there is a dual problem

$$\begin{array}{ll} \text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x + c^f \\ & A^T y + s_l^x - s_u^x = c, \\ \text{subject to} & -y + s_l^c - s_u^c = 0, \\ & s_l^c, s_u^c, s_l^x, s_u^x \geq 0. \end{array} \quad (12.2)$$

If a bound in the primal problem is plus or minus infinity, the corresponding dual variable is fixed at 0, and we use the convention that the product of the bound value and the corresponding dual variable is 0. 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 (12.2). If (12.2) has at least one feasible solution, then (12.2) 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 (12.1) and $(y, s_l^c, s_u^c, s_l^x, s_u^x)$ is a solution to the corresponding dual problem (12.2).

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,

$$\begin{aligned} c^T x^* + c^f - \{ & (l^c)^T (s_l^c)^* - (u^c)^T (s_u^c)^* + (l^x)^T (s_l^x)^* - (u^x)^T (s_u^x)^* + c^f \} \\ &= \sum_{i=0}^{m-1} [(s_l^c)^* ((x_i^c)^* - l_i^c) + (s_u^c)^* (u_i^c - (x_i^c)^*)] \\ &+ \sum_{j=0}^{n-1} [(s_l^x)^* (x_j - l_j^x) + (s_u^x)^* (u_j^x - x_j^*)] \geq 0 \end{aligned} \quad (12.3)$$

where the first relation can be obtained by transposing and multiplying the dual constraints (12.2) by x^* and $(x^c)^*$ respectively, and the second relation comes from the fact that each term in each sum is nonnegative. It follows that the primal objective will always be greater than or equal to the dual objective.

An Optimal Solution

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

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

are satisfied.

If (12.1) has an optimal solution and **MOSEK** solves the problem successfully, both the primal and dual solution are reported, including a status indicating the exact state of the solution.

12.1.2 Infeasibility for Linear Optimization

Primal Infeasible Problems

If the problem (12.1) is infeasible (has no feasible solution), **MOSEK** will report a certificate of primal infeasibility: The dual solution reported is the certificate of infeasibility, and the primal solution is undefined.

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

$$\begin{aligned} & \text{maximize} && (l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x \\ & \text{subject to} && \\ & && A^T y + s_l^x - s_u^x = 0, \\ & && -y + s_l^c - s_u^c = 0, \\ & && s_l^c, s_u^c, s_l^x, s_u^x \geq 0, \end{aligned} \tag{12.4}$$

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

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

to (12.4) so that

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

Such a solution implies that (12.4) is unbounded, and that its dual is infeasible. As the constraints to the dual of (12.4) are identical to the constraints of problem (12.1), we thus have that problem (12.1) is also infeasible.

Dual Infeasible Problems

If the problem (12.2) is infeasible (has no feasible solution), **MOSEK** will report a certificate of dual infeasibility: The primal solution reported is the certificate of infeasibility, and the dual solution is undefined.

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

$$\begin{aligned} & \text{minimize} && c^T x \\ & \text{subject to} && \hat{l}^c \leq Ax \leq \hat{u}^c, \\ & && \hat{l}^x \leq x \leq \hat{u}^x, \end{aligned} \tag{12.5}$$

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

$$c^T x < 0.$$

Such a solution implies that (12.5) is unbounded, and that its dual is infeasible. As the constraints to the dual of (12.5) are identical to the constraints of problem (12.2), we thus have that problem (12.2) is also infeasible.

Primal and Dual Infeasible Case

In case that both the primal problem (12.1) and the dual problem (12.2) are infeasible, **MOSEK** will report only one of the two possible certificates — which one is not defined (**MOSEK** returns the first certificate found).

Minimalization vs. Maximalization

When the objective sense of problem (12.1) is maximization, i.e.

$$\begin{array}{llll} \text{maximize} & & c^T x + c^f \\ \text{subject to} & l^c \leq & Ax & \leq u^c, \\ & l^x \leq & x & \leq u^x, \end{array}$$

the objective sense of the dual problem changes to minimization, and the domain of all dual variables changes sign in comparison to (12.2). The dual problem thus takes the form

$$\begin{array}{ll} \text{minimize} & (l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x + c^f \\ \text{subject to} & \\ & A^T y + s_l^x - s_u^x = c, \\ & -y + s_l^c - s_u^c = 0, \\ & s_l^c, s_u^c, s_l^x, s_u^x \leq 0. \end{array}$$

This means that the duality gap, defined in (12.3) as the primal minus the dual objective value, becomes nonpositive. It follows that the dual objective will always be greater than or equal to the primal objective. The primal infeasibility certificate will be reported by **MOSEK** as a solution to the system

$$\begin{array}{l} A^T y + s_l^x - s_u^x = 0, \\ -y + s_l^c - s_u^c = 0, \\ s_l^c, s_u^c, s_l^x, s_u^x \leq 0, \end{array} \quad (12.6)$$

such that the objective value is strictly negative

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

Similarly, the certificate of dual infeasibility is an x satisfying the requirements of (12.5) such that $c^T x > 0$.

12.2 Conic Quadratic Optimization

Conic quadratic optimization is an extension of linear optimization (see Section 12.1) allowing conic domains to be specified for subsets of the problem variables. A conic quadratic optimization problem can be written as

$$\begin{array}{llll} \text{minimize} & & c^T x + c^f \\ \text{subject to} & l^c \leq & Ax & \leq u^c, \\ & l^x \leq & x & \leq u^x, \\ & & x \in \mathcal{K}, \end{array} \quad (12.7)$$

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 $x = (x^1, \dots, x^p)$ 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:

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

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 [\[MOSEKApS12\]](#).

12.2.1 Duality for Conic Quadratic Optimization

The dual problem corresponding to the conic quadratic optimization problem (12.7) is given by

$$\begin{aligned} & \text{maximize} && (l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x + c^f \\ & \text{subject to} && \\ & && A^T y + s_l^x - s_u^x + s_n^x = c \\ & && -y + s_l^c - s_u^c = 0, \\ & && s_l^c, s_u^c, s_l^x, s_u^x \geq 0, \\ & && s_n^x \in \mathcal{K}^*, \end{aligned} \tag{12.8}$$

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:

$$\mathcal{K}_t = \mathbb{R}^{n_t} \quad \Leftrightarrow \quad \mathcal{K}_t^* = \{s \in \mathbb{R}^{n_t} : s = 0\}.$$

- 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.

12.2.2 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 12.1.2).

Primal Infeasible Problems

If the problem (12.7) 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

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

such that the objective value is strictly positive.

Dual infeasible problems

If the problem (12.8) 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

$$\begin{aligned} & \text{minimize} && c^T x \\ & \text{subject to} && \hat{l}^c \leq Ax \leq \hat{u}^c, \\ & && \hat{l}^x \leq x \leq \hat{u}^x, \\ & && x \in \mathcal{K}, \end{aligned}$$

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.

12.3 Semidefinite Optimization

Semidefinite optimization is an extension of conic quadratic optimization (see Section 12.2) allowing positive semidefinite matrix variables to be used in addition to the usual scalar variables. A semidefinite optimization problem can be written as

$$\begin{aligned} & \text{minimize} && \sum_{j=0}^{n-1} c_j x_j + \sum_{j=0}^{p-1} \langle \bar{C}_j, \bar{X}_j \rangle + c^f \\ & \text{subject to} && \begin{aligned} l_i^c &\leq && \sum_{j=0}^{n-1} a_{ij} x_j + \sum_{j=0}^{p-1} \langle \bar{A}_{ij}, \bar{X}_j \rangle &\leq & u_i^c, & i = 0, \dots, m-1 \\ l_j^x &\leq && x_j &\leq & u_j^x, & j = 0, \dots, n-1 \\ &&& x \in \mathcal{K}, \bar{X}_j \in \mathcal{S}_+^{r_j}, && j = 0, \dots, p-1 \end{aligned} \end{aligned} \quad (12.9)$$

where the problem has p symmetric positive semidefinite variables $\bar{X}_j \in \mathcal{S}_+^{r_j}$ of dimension r_j with symmetric coefficient matrices $\bar{C}_j \in \mathcal{S}^{r_j}$ and $\bar{A}_{ij} \in \mathcal{S}^{r_j}$. 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].

12.3.1 Duality for Semidefinite Optimization

The dual problem corresponding to the semidefinite optimization problem (12.9) is given by

$$\begin{aligned} & \text{maximize} && (l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x + c^f \\ & \text{subject to} && \begin{aligned} c - A^T y + s_u^x - s_l^x &= s_n^x, \\ \bar{C}_j - \sum_{i=0}^m y_i \bar{A}_{ij} &= \bar{S}_j, && j = 0, \dots, p-1 \\ s_l^c - s_u^c &= y, \\ s_l^c, s_u^c, s_l^x, s_u^x &\geq 0, \\ s_n^x &\in \mathcal{K}^*, \quad \bar{S}_j \in \mathcal{S}_+^{r_j}, && j = 0, \dots, p-1 \end{aligned} \end{aligned} \quad (12.10)$$

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 12.2.1), except for the addition of dual constraints

$$\left(\bar{C}_j - \sum_{i=0}^m y_i \bar{A}_{ij} \right) \in \mathcal{S}_+^{r_j}.$$

Note that the dual of the dual problem is identical to the original primal problem.

12.3.2 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 12.1.2).

Primal Infeasible Problems

If the problem (12.9) 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

$$\begin{aligned} & \text{maximize} && (l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x \\ & \text{subject to} && \\ & && A^T y + s_l^x - s_u^x + s_n^x = 0, \\ & && \sum_{i=0}^{m-1} y_i \bar{A}_{ij} + \bar{S}_j = 0, && j = 0, \dots, p-1 \\ & && -y + s_l^c - s_u^c = 0, \\ & && s_l^c, s_u^c, s_l^x, s_u^x \geq 0, \\ & && s_n^x \in \mathcal{K}^*, \quad \bar{S}_j \in \mathcal{S}_+^{r_j}, && j = 0, \dots, p-1 \end{aligned}$$

such that the objective value is strictly positive.

Dual Infeasible Problems

If the problem (12.10) 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

$$\begin{aligned} & \text{minimize} && \sum_{j=0}^{n-1} c_j x_j + \sum_{j=0}^{p-1} \langle \bar{C}_j, \bar{X}_j \rangle \\ & \text{subject to} && \hat{l}_i^c \leq \sum_{j=1}^n a_{ij} x_j + \sum_{j=0}^{p-1} \langle \bar{A}_{ij}, \bar{X}_j \rangle \leq \hat{u}_i^c, \quad i = 0, \dots, m-1 \\ & && \hat{l}^x \leq \begin{matrix} x \\ \bar{X}_j \end{matrix} \leq \hat{u}^x, \\ & && x \in \mathcal{K}, \quad \bar{X}_j \in \mathcal{S}_+^{r_j}, && j = 0, \dots, p-1 \end{aligned}$$

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.

12.4 Quadratic and Quadratically Constrained Optimization

A convex quadratic and quadratically constrained optimization problem is an optimization problem of the form

$$\begin{aligned} & \text{minimize} && \frac{1}{2}x^T Q^o x + c^T x + c^f \\ & \text{subject to} && \begin{aligned} l_k^c &\leq \frac{1}{2}x^T Q^k x + \sum_{j=0}^{n-1} a_{kj} x_j &\leq u_k^c, & k = 0, \dots, m-1, \\ l_j^x &\leq x_j &\leq u_j^x, & j = 0, \dots, n-1, \end{aligned} \end{aligned} \quad (12.11)$$

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

$$\begin{aligned} -\infty < l_k^c &\Rightarrow Q^k \text{ is negative semidefinite,} \\ u_k^c < \infty &\Rightarrow Q^k \text{ is positive semidefinite,} \\ -\infty < l_k^c \leq u_k^c < \infty &\Rightarrow Q^k = 0. \end{aligned}$$

The convexity requirement is very important and **MOSEK** checks whether it is fulfilled.

12.4.1 A Recommendation

Any convex quadratic optimization problem can be reformulated as a conic quadratic optimization problem, see [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 formulation if deemed worthwhile potentially leading to (huge) computational savings.

However, instead of relying on the automatic reformulation we recommend to formulate the problem as conic problem from scratch because:

- it saves the computational overhead of the reformulation including the convexity check. A conic problem is convex by construction and hence no convexity check is needed for conic problems.
- usually the 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.

12.4.2 Duality for Quadratic and Quadratically Constrained Optimization

The dual problem corresponding to the quadratic and quadratically constrained optimization problem (12.11) is given by

$$\begin{aligned} & \text{maximize} && (l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x + \frac{1}{2}x^T \left\{ \sum_{k=0}^{m-1} y_k Q^k - Q^o \right\} x + c^f \\ & \text{subject to} && \begin{aligned} A^T y + s_l^x - s_u^x + \left\{ \sum_{k=0}^{m-1} y_k Q^k - Q^o \right\} x &= c, \\ -y + s_l^c - s_u^c &= 0, \\ s_l^c, s_u^c, s_l^x, s_u^x &\geq 0. \end{aligned} \end{aligned} \quad (12.12)$$

The dual problem is related to the dual problem for linear optimization (see Section 12.1.1), but depends on the variable x which in general can not be eliminated. In the solutions reported by **MOSEK**, the value of x is the same for the primal problem (12.11) and the dual problem (12.12).

12.4.3 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 12.1.2).

Primal Infeasible Problems

If the problem (12.11) 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 12.1.2.1).

Dual Infeasible Problems

If the problem (12.12) 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

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & \hat{l}^c \leq Ax \leq \hat{u}^c, \\ & 0 \leq Q^o x \leq 0, \\ & \hat{l}^x \leq x \leq \hat{u}^x, \end{array}$$

where

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

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.

12.5 General Convex Optimization

MOSEK is capable of solving smooth (twice differentiable) convex nonlinear optimization problems of the form

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

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.

- $u^x \in \mathbb{R}^n$ is the upper limit on the activity for the variables.
- $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is a nonlinear function.
- $g : \mathbb{R}^n \rightarrow \mathbb{R}^m$ 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

$$\begin{aligned} -\infty < l_i^c &\Rightarrow g_i(x) \text{ is concave,} \\ u_i^c < \infty &\Rightarrow g_i(x) \text{ is convex,} \\ -\infty < l_i^c \leq u_i^c < \infty &\Rightarrow g_i(x) = 0. \end{aligned}$$

12.5.1 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

$$\begin{aligned} L(x, s_l^c, s_u^c, s_l^x, s_u^x) &:= f(x) + c^T x + c^f \\ &\quad - (s_l^c)^T (g(x) + Ax - l^c) - (s_u^c)^T (u^c - g(x) - Ax) \\ &\quad - (s_l^x)^T (x - l^x) - (s_u^x)^T (u^x - x), \end{aligned}$$

and the dual problem is given by

$$\begin{aligned} &\text{maximize} && L(x, s_l^c, s_u^c, s_l^x, s_u^x) \\ &\text{subject to} && \nabla_x L(x, s_l^c, s_u^c, s_l^x, s_u^x)^T = 0, \\ &&& s_l^c, s_u^c, s_l^x, s_u^x \geq 0, \end{aligned}$$

which is equivalent to

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

In this context we use the following definition for scalar functions

$$\nabla f(x) = \left[\frac{\partial f(x)}{\partial x_1}, \dots, \frac{\partial f(x)}{\partial x_n} \right],$$

and accordingly for vector functions

$$\nabla g(x) = \begin{bmatrix} \nabla g_1(x) \\ \vdots \\ \nabla g_m(x) \end{bmatrix}.$$

THE OPTIMIZERS FOR CONTINUOUS PROBLEMS

The most essential part of **MOSEK** are the optimizers. This chapter describes the optimizers for the class of *continuous problems* without integer variables, that is:

- linear problems,
- conic problems (quadratic and semidefinite),
- general convex problems.

MOSEK offers an interior-point optimizer for each class of problems and also a simplex optimizer for linear problems. The structure of a successful optimization process is roughly:

- **Presolve**
 1. *Elimination*: Reduce the size of the problem.
 2. *Dualizer*: Choose whether to solve the primal or the dual form of the problem.
 3. *Scaling*: Scale the problem for better numerical stability.
- **Optimization**
 1. *Optimize*: Solve the problem using selected method.
 2. *Terminate*: Stop the optimization when specific termination criteria have been met.
 3. *Report*: Return the solution or an infeasibility certificate.

The preprocessing stage is transparent to the user, but useful to know about for tuning purposes. The purpose of the preprocessing steps is to make the actual optimization more efficient and robust. We discuss the details of the above steps in the following sections.

13.1 Presolve

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

1. remove redundant constraints,
2. eliminate fixed variables,
3. remove linear dependencies,
4. substitute out (implied) free variables, and
5. reduce the size of the optimization problem in general.

After the presolved problem has been optimized the solution is automatically postsolved so that the returned solution is valid for the original problem. Hence, the presolve is completely transparent. For further details about the presolve phase, please see [\[AA95\]](#) and [\[AGMX96\]](#).

It is possible to fine-tune the behavior of the presolve or to turn it off entirely. If presolve consumes too much time or memory compared to the reduction in problem size gained it may be disabled. This is done

by setting the parameter `iparam.presolve_use` to `presolvemode.off`. The two most time-consuming steps of the presolve are

- the eliminator, and
- the linear dependency check.

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

Numerical issues in the presolve

During the presolve the problem is reformulated so that it hopefully solves faster. However, in rare cases the presolved problem may be harder to solve than the original problem. The presolve may also be infeasible although the original problem is not. If it is suspected that presolved problem is much harder to solve than the original, we suggest to first turn the eliminator off by setting the parameter `iparam.presolve_eliminator_max_num_tries` to 0. If that does not help, then trying to turn entire presolve off may help.

Since all computations are done in finite precision, the presolve employs some tolerances when concluding a variable is fixed or a constraint is redundant. If it happens that **MOSEK** incorrectly concludes a problem is primal or dual infeasible, then it is worthwhile to try to reduce the parameters `dparam.presolve_tol_x` and `dparam.presolve_tol_s`. However, if reducing the parameters actually helps then this should be taken as an indication that the problem is badly formulated.

Eliminator

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

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

y is an implied free variable that can be substituted out of the problem, if deemed worthwhile. If the eliminator consumes too much time or memory compared to the reduction in problem size gained it may be disabled. This can be done 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

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

contain exactly one linear dependency. This implies that one of the constraints can be dropped without changing the set of feasible solutions. Removing linear dependencies is in general a good idea since it reduces the size of the problem. Moreover, the linear dependencies are likely to introduce numerical problems in the optimization phase. It is best practice to build models without linear dependencies, but that is not always easy for the user to control. If the linear dependencies are removed at the modelling stage, the linear dependency check can safely be disabled by setting the parameter `iparam.presolve_linddep_use` to `onoffkey.off`.

Dualizer

All linear, conic, and convex optimization problems have an equivalent dual problem associated with them. **MOSEK** has built-in heuristics to determine if it is more efficient to solve the primal or dual

problem. The form (primal or dual) is displayed in the **MOSEK** log and available as an information item from the solver. Should the internal heuristics not choose the most efficient form of the problem it may be worthwhile to set the dualizer manually by setting the parameters:

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

Note that currently only linear and conic quadratic problems may be automatically dualized.

Scaling

Problems containing data with large and/or small coefficients, say $1.0e + 9$ or $1.0e - 7$, are often hard to solve. Significant digits may be truncated in calculations with finite precision, which can result in the optimizer relying on inaccurate data. Since computers work in finite precision, extreme coefficients should be avoided. In general, data around the same *order of magnitude* is preferred, and we will refer to a problem, satisfying this loose property, as being *well-scaled*. If the problem is not well scaled, **MOSEK** will try to scale (multiply) constraints and variables by suitable constants. **MOSEK** solves the scaled problem to improve the numerical properties.

The scaling process is transparent, i.e. the solution to the original problem is reported. It is important to be aware that the optimizer terminates when the termination criterion is met on the scaled problem, therefore significant primal or dual infeasibilities may occur after unscaling for badly scaled problems. The best solution of this issue is to reformulate the problem, making it better scaled.

By default **MOSEK** heuristically chooses a suitable scaling. The scaling for interior-point and simplex optimizers can be controlled with the parameters `iparam.intpnt_scaling` and `iparam.sim_scaling` respectively.

13.2 Using Multiple Threads in an Optimizer

Multithreading in interior-point optimizers

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 maximum 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 because of the additional coordination overhead. Therefore, it may be advantageous to disable multithreading using the parameter `iparam.intpnt_multi_thread`.

The interior-point optimizer parallelizes big tasks such linear algebra computations.

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.

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 restriction 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.

13.3 Linear Optimization

13.3.1 Optimizer Selection

Two different types of optimizers are available for linear problems: The default is an interior-point method, and the alternative is the simplex method (primal or dual). The optimizer can be selected using the parameter *iparam.optimizer*.

The Interior-point or the Simplex Optimizer?

Given a linear optimization problem, which optimizer is the best: the simplex or the interior-point optimizer? It is impossible to provide a general answer to this question. However, the interior-point optimizer behaves more predictably: it tends to use between 20 and 100 iterations, almost independently of problem size, but cannot perform warm-start. On the other hand the simplex method can take advantage of an initial solution, but is less predictable from cold-start. The interior-point optimizer is used by default.

The Primal or the Dual Simplex Variant?

MOSEK provides both a primal and a dual simplex optimizer. Predicting which simplex optimizer is faster is impossible, however, in recent years the dual optimizer has seen several algorithmic and computational improvements, which, in our experience, make it faster on average than the primal version. Still, it depends much on the problem structure and size. Setting the *iparam.optimizer* parameter to *optimizertype.free_simplex* instructs **MOSEK** to choose one of the simplex variants automatically.

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

13.3.2 The Interior-point Optimizer

The purpose of this section is to provide information about the algorithm employed in the **MOSEK** interior-point optimizer for linear problems and about its termination criteria.

The homogeneous primal-dual problem

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

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax = b, \\ & x \geq 0. \end{array} \quad (13.1)$$

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

Since it is not known beforehand whether problem (13.1) has an optimal solution, is primal infeasible or is dual infeasible, the optimization algorithm must deal with all three situations. This is the reason why **MOSEK** solves the so-called homogeneous model

$$\begin{aligned} Ax - b\tau &= 0, \\ A^T y + s - c\tau &= 0, \\ -c^T x + b^T y - \kappa &= 0, \\ x, s, \tau, \kappa &\geq 0, \end{aligned} \tag{13.2}$$

where y and s correspond to the dual variables in (13.1), and τ and κ are two additional scalar variables. Note that the homogeneous model (13.2) always has solution since

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

is a solution, although not a very interesting one. Any solution

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

to the homogeneous model (13.2) satisfies

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

Moreover, there is always a solution that has the property $\tau^* + \kappa^* > 0$.

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

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

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

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

is a primal-dual optimal solution (see Section 12.1 for the mathematical background on duality and optimality).

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

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

This implies that at least one of

$$c^T x^* < 0 \tag{13.3}$$

or

$$b^T y^* > 0 \tag{13.4}$$

is satisfied. If (13.3) is satisfied then x^* is a certificate of dual infeasibility, whereas if (13.4) is satisfied then y^* is a certificate of primal infeasibility.

In summary, by computing an appropriate solution to the homogeneous model, all information required for a solution to the original problem is obtained. A solution to the homogeneous model can be computed using a primal-dual interior-point algorithm [And09].

Interior-point Termination Criterion

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

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

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

to homogeneous model is generated, where

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

Optimal case

Whenever the trial solution satisfies the criterion

$$\begin{aligned} \left\| A \frac{x^k}{\tau^k} - b \right\|_{\infty} &\leq \epsilon_p (1 + \|b\|_{\infty}), \\ \left\| A^T \frac{y^k}{\tau^k} + \frac{s^k}{\tau^k} - c \right\|_{\infty} &\leq \epsilon_d (1 + \|c\|_{\infty}), \text{ and} \\ \min \left(\frac{(x^k)^T s^k}{(\tau^k)^2}, \left| \frac{c^T x^k}{\tau^k} - \frac{b^T y^k}{\tau^k} \right| \right) &\leq \epsilon_g \max \left(1, \frac{\min(|c^T x^k|, |b^T y^k|)}{\tau^k} \right), \end{aligned} \quad (13.5)$$

the interior-point optimizer is terminated and

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

is reported as the primal-dual optimal solution. The interpretation of (13.5) is that the optimizer is terminated if

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

Dual infeasibility certificate

On the other hand, if the trial solution satisfies

$$-\epsilon_i c^T x^k > \frac{\|c\|_{\infty}}{\max(1, \|b\|_{\infty})} \|Ax^k\|_{\infty}$$

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 $\|Ax^k\|_{\infty} = 0$; then x^k is an exact certificate of dual infeasibility. Next assume that this is not the case, i.e.

$$\|Ax^k\|_{\infty} > 0,$$

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.

Primal infeasibility certificate

Finally, if

$$\epsilon_i b^T y^k > \frac{\|b\|_\infty}{\max(1, \|c\|_\infty)} \|A^T y^k + s^k\|_\infty$$

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

Adjusting optimality criteria and near optimality

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

Table 13.1: Parameters employed in termination criterion

ToleranceParameter	name
ϵ_p	<code>dparam.intpnt_tol_pfeas</code>
ϵ_d	<code>dparam.intpnt_tol_dfeas</code>
ϵ_g	<code>dparam.intpnt_tol_rel_gap</code>
ϵ_i	<code>dparam.intpnt_tol_infeas</code>

The default values of the termination tolerances are chosen such that for a majority of problems appearing in practice it is not possible to achieve much better accuracy. Therefore, tightening the tolerances usually is not worthwhile. However, an inspection of (13.5) reveals that the quality of the solution depends on $\|b\|_\infty$ and $\|c\|_\infty$; the smaller the norms are, the better the solution accuracy.

The interior-point method as implemented by **MOSEK** will converge toward optimality and primal and dual feasibility at the same rate [And09]. This means that if the optimizer is stopped prematurely then it is very unlikely that either the primal or dual solution is feasible. Another consequence is that in most cases all the tolerances, ϵ_p , ϵ_d , ϵ_g and ϵ_i , have to be relaxed together to achieve an effect.

In some cases the interior-point method terminates having found a solution not too far from meeting the optimality condition (13.5). A solution is defined as *near optimal* if scaling the termination tolerances ϵ_p , ϵ_d , ϵ_g and ϵ_i by the same factor $\epsilon_n \in [1.0, +\infty]$ makes the condition (13.5) 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. Near infeasibility certificates are defined similarly. The value of ϵ_n can be adjusted with the parameter `dparam.intpnt_co_tol_near_rel`.

The basis identification discussed in Section 13.3.2.4 requires an optimal solution to work well; hence basis identification should be turned off if the termination criterion is relaxed.

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

Basis Identification

An interior-point optimizer does not return an optimal basic solution unless the problem has a unique primal and dual optimal solution. Therefore, the interior-point optimizer has an optional post-processing step that computes an optimal basic solution starting from the optimal interior-point solution. More information about the basis identification procedure may be found in [AY96]. In the following we provide an overall idea of the procedure.

There are some cases in which a basic solution could be more valuable:

- a basic solution is often more accurate than an interior-point solution,
- a basic solution can be used to warm-start the simplex algorithm in case of reoptimization,
- a basic solution is in general more sparse, i.e. more variables are fixed to zero. This is particularly appealing when solving continuous relaxations of mixed integer problems, as well as in all applications in which sparser solutions are preferred.

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 13.3.2 the columns of the iteration log have the following meaning:

- **ITE**: Iteration index k .
- **PFEAS**: $\|Ax^k - b\tau^k\|_\infty$. The numbers in this column should converge monotonically towards zero but may stall at low level due to rounding errors.
- **DFEAS**: $\|A^T y^k + s^k - c\tau^k\|_\infty$. The numbers in this column should converge monotonically towards zero but may stall at low level due to rounding errors.
- **GFEAS**: $|-c^T x^k + b^T y^k - \kappa^k|$. The numbers in this column should converge monotonically towards zero but may stall at low level due to rounding errors.
- **PRSTATUS**: This number converges to 1 if the problem has an optimal solution whereas it converges to -1 if that is not the case.
- **POBJ**: $c^T x^k / \tau^k$. An estimate for the primal objective value.
- **DOBJ**: $b^T y^k / \tau^k$. An estimate for the dual objective value.
- **MU**: $\frac{(x^k)^T s^k + \tau^k \kappa^k}{n+1}$. The numbers in this column should always converge to zero.
- **TIME**: Time spent since the optimization started.

13.3.3 The Simplex Optimizer

An alternative to the interior-point optimizer is the simplex optimizer. The simplex optimizer uses a different method that allows exploiting an initial guess for the optimal solution to reduce the solution time. Depending on the problem it may be faster or slower to use an initial guess; see section 13.3.1 for a discussion. **MOSEK** provides both a primal and a dual variant of the simplex optimizer.

Simplex Termination Criterion

The simplex optimizer terminates when it finds an optimal basic solution or an infeasibility certificate. A basic solution is optimal when it is primal and dual feasible; see Section 12.1 for a definition of the primal and dual problem. Due to the fact that computations are performed in finite precision **MOSEK** allows violations of primal and dual feasibility within certain tolerances. The user can control the allowed primal and dual tolerances with the parameters `dparam.basis_tol_x` and `dparam.basis_tol_s`.

Setting the parameter `iparam.optimizer` to `optimizertype.free_simplex` instructs **MOSEK** to select automatically between the primal and the dual simplex optimizers. Hence, **MOSEK** tries to choose the best optimizer for the given problem and the available solution. The same parameter can also be used to force one of the variants.

Starting From an Existing Solution

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

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

Numerical Difficulties in the Simplex Optimizers

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

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

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

The Simplex Log

Below is a typical log output from the simplex optimizer:

Optimizer	- solved problem	:	the primal			
Optimizer	- Constraints	:	667			
Optimizer	- Scalar variables	:	1424	conic	:	0
Optimizer	- hotstart	:	no			
ITER	DEGITER(%)	PFEAS	DFEAS	POBJ	DOBJ	TIME
↪	TOTTIME					
0	0.00	1.43e+05	NA	6.5584140832e+03	NA	0.00
↪	0.02					
1000	1.10	0.00e+00	NA	1.4588289726e+04	NA	0.13
↪	0.14					
2000	0.75	0.00e+00	NA	7.3705564855e+03	NA	0.21
↪	0.22					
3000	0.67	0.00e+00	NA	6.0509727712e+03	NA	0.29
↪	0.31					
4000	0.52	0.00e+00	NA	5.5771203906e+03	NA	0.38
↪	0.39					
4533	0.49	0.00e+00	NA	5.5018458883e+03	NA	0.42
↪	0.44					

The first lines summarize the problem the optimizer is solving. This is followed by the iteration log, with the following meaning:

- ITER: Number of iterations.
- DEGITER(%): Ratio of degenerate iterations.
- PFEAS: Primal feasibility measure reported by the simplex optimizer. The numbers should be 0 if the problem is primal feasible (when the primal variant is used).
- DFEAS: Dual feasibility measure reported by the simplex optimizer. The number should be 0 if the problem is dual feasible (when the dual variant is used).
- POBJ: An estimate for the primal objective value (when the primal variant is used).
- DOBJ: An estimate for the dual objective value (when the dual variant is used).
- TIME: Time spent since this instance of the simplex optimizer was invoked (in seconds).
- TOTTIME: Time spent since optimization started (in seconds).

13.4 Conic Optimization

For conic optimization problems only an interior-point type optimizer is available.

13.4.1 The Interior-point optimizer

The homogeneous primal-dual problem

The interior-point optimizer is an implementation of the so-called homogeneous and self-dual algorithm. For a detailed description of the algorithm, please see [\[ART03\]](#). In order to keep our discussion simple we will assume that **MOSEK** solves a conic optimization problem of the form:

$$\begin{aligned} & \text{minimize} && c^T x \\ & \text{subject to} && Ax = b, \\ & && x \in \mathcal{K} \end{aligned} \tag{13.6}$$

where \mathcal{K} is a convex cone. The corresponding dual problem is

$$\begin{aligned} & \text{maximize} && b^T y \\ & \text{subject to} && A^T y + s = c, \\ & && x \in \mathcal{K}^* \end{aligned} \tag{13.7}$$

where \mathcal{K}^* is the dual cone of \mathcal{K} . See Section 12.2 for definitions.

Since it is not known beforehand whether problem (13.6) has an optimal solution, is primal infeasible or is dual infeasible, the optimization algorithm must deal with all three situations. This is the reason that **MOSEK** solves the so-called homogeneous model

$$\begin{aligned} Ax - b\tau &= 0, \\ A^T y + s - c\tau &= 0, \\ -c^T x + b^T y - \kappa &= 0, \\ x &\in \mathcal{K}, \\ s &\in \mathcal{K}^*, \\ \tau, \kappa &\geq 0, \end{aligned} \tag{13.8}$$

where y and s correspond to the dual variables in (13.6), and τ and κ are two additional scalar variables. Note that the homogeneous model (13.8) always has a solution since

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

is a solution, although not a very interesting one. Any solution

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

to the homogeneous model (13.8) satisfies

$$(x^*)^T s^* + \tau^* \kappa^* = 0$$

i.e. complementarity. Observe that $x^* \in \mathcal{K}$ and $s^* \in \mathcal{K}^*$ implies

$$(x^*)^T s^* \geq 0$$

and therefore

$$\tau^* \kappa^* = 0.$$

since $\tau^*, \kappa^* \geq 0$. Hence, at least one of τ^* and κ^* is zero.

First, assume that $\tau^* > 0$ and hence $\kappa^* = 0$. It follows that

$$\begin{aligned} A \frac{x^*}{\tau^*} &= b, \\ A^T \frac{y^*}{\tau^*} + \frac{s^*}{\tau^*} &= c, \\ -c^T \frac{x^*}{\tau^*} + b^T \frac{y^*}{\tau^*} &= 0, \\ x^*/\tau^* &\in \mathcal{K}, \\ s^*/\tau^* &\in \mathcal{K}^*. \end{aligned}$$

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

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

is a primal-dual optimal solution.

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

$$\begin{aligned} Ax^* &= 0, \\ A^T y^* + s^* &= 0, \\ -c^T x^* + b^T y^* &= \kappa^*, \\ x^* &\in \mathcal{K}, \\ s^* &\in \mathcal{K}^*. \end{aligned}$$

This implies that at least one of

$$c^T x^* < 0 \tag{13.9}$$

or

$$b^T y^* > 0 \tag{13.10}$$

holds. If (13.9) is satisfied, then x^* is a certificate of dual infeasibility, whereas if (13.10) holds then y^* is a certificate of primal infeasibility.

In summary, by computing an appropriate solution to the homogeneous model, all information required for a solution to the original problem is obtained. A solution to the homogeneous model can be computed using a primal-dual interior-point algorithm [And09].

Interior-point Termination Criterion

Since computations are performed in finite precision, and for efficiency reasons, it is not possible to solve the homogeneous model exactly in general. Hence, an exact optimal solution or an exact infeasibility certificate cannot be computed and a reasonable termination criterion has to be employed.

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

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

to the homogeneous model is generated, where

$$x^k \in \mathcal{K}, s^k \in \mathcal{K}^*, \tau^k, \kappa^k > 0.$$

Therefore, it is possible to compute the values:

$$\begin{aligned} \rho_p^k &= \arg \min_{\rho} \left\{ \rho \mid \left\| A \frac{x^k}{\tau^k} - b \right\|_{\infty} \leq \rho \varepsilon_p (1 + \|b\|_{\infty}) \right\}, \\ \rho_d^k &= \arg \min_{\rho} \left\{ \rho \mid \left\| A^T \frac{y^k}{\tau^k} + \frac{s^k}{\tau^k} - c \right\|_{\infty} \leq \rho \varepsilon_d (1 + \|c\|_{\infty}) \right\}, \\ \rho_g^k &= \arg \min_{\rho} \left\{ \rho \mid \left(\frac{(x^k)^T s^k}{(\tau^k)^2}, \left| \frac{c^T x^k}{\tau^k} - \frac{b^T y^k}{\tau^k} \right| \right) \leq \rho \varepsilon_g \max \left(1, \frac{\min(|c^T x^k|, |b^T y^k|)}{\tau^k} \right) \right\}, \\ \rho_{pi}^k &= \arg \min_{\rho} \left\{ \rho \mid \left\| A^T y^k + s^k \right\|_{\infty} \leq \rho \varepsilon_i b^T y^k, b^T y^k > 0 \right\} \text{ and} \\ \rho_{di}^k &= \arg \min_{\rho} \left\{ \rho \mid \left\| Ax^k \right\|_{\infty} \leq -\rho \varepsilon_i c^T x^k, c^T x^k < 0 \right\}. \end{aligned}$$

Note $\varepsilon_p, \varepsilon_d, \varepsilon_g$ and ε_i are nonnegative user specified tolerances.

Optimal Case

Observe ρ_p^k measures how far x^k/τ^k is from being a good approximate primal feasible solution. Indeed if $\rho_p^k \leq 1$, then

$$\left\| A \frac{x^k}{\tau^k} - b \right\|_{\infty} \leq \varepsilon_p (1 + \|b\|_{\infty}). \quad (13.11)$$

This shows the violations in the primal equality constraints for the solution x^k/τ^k is small compared to the size of b given ε_p is small.

Similarly, if $\rho_d^k \leq 1$, then $(y^k, s^k)/\tau^k$ is an approximate dual feasible solution. If in addition $\rho_g^k \leq 1$, then the solution $(x^k, y^k, s^k)/\tau^k$ is approximate optimal because the associated primal and dual objective values are almost identical.

In other words if $\max(\rho_p^k, \rho_d^k, \rho_g^k) \leq 1$, then

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

is an approximate optimal solution.

Dual Infeasibility Certificate

Next assume that $\rho_{di}^k \leq 1$ and hence

$$\|Ax^k\|_{\infty} \leq -\varepsilon_i c^T x^k \text{ and } -c^T x^k > 0$$

holds. Now in this case the problem is declared dual infeasible and x^k is reported as a certificate of dual infeasibility. The motivation for this stopping criterion is as follows. Let

$$\bar{x} := \frac{x^k}{-c^T x^k}$$

and it is easy to verify that

$$\|A\bar{x}\|_{\infty} \leq \varepsilon_i \text{ and } c^T \bar{x} = -1$$

which shows \bar{x} is an approximate certificate of dual infeasibility, where ε_i controls the quality of the approximation.

Primal Infeasibility Certificate

Next assume that $\rho_{pi}^k \leq 1$ and hence

$$\|A^T y^k + s^k\|_\infty \leq \varepsilon_i b^T y^k \text{ and } b^T y^k > 0$$

holds. Now in this case the problem is declared primal infeasible and (y^k, s^k) is reported as a certificate of primal infeasibility. The motivation for this stopping criterion is as follows. Let

$$\bar{y} := \frac{y^k}{b^T y^k} \text{ and } \bar{s} := \frac{s^k}{b^T y^k}$$

and it is easy to verify that

$$\|A^T \bar{y} + \bar{s}\|_\infty \leq \varepsilon_i \text{ and } b^T \bar{y} = 1$$

which shows (y^k, s^k) is an approximate certificate of dual infeasibility, where ε_i controls the quality of the approximation.

Adjusting optimality criteria and near optimality

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

Table 13.2: Parameters employed in termination criterion

ToleranceParameter	name
ε_p	<code>dparam.intpnt_tol_pfeas</code>
ε_d	<code>dparam.intpnt_tol_dfeas</code>
ε_g	<code>dparam.intpnt_tol_rel_gap</code>
ε_i	<code>dparam.intpnt_tol_infeas</code>

The default values of the termination tolerances are chosen such that for a majority of problems appearing in practice it is not possible to achieve much better accuracy. Therefore, tightening the tolerances usually is not worthwhile. However, an inspection of (13.11) reveals that the quality of the solution depends on $\|b\|_\infty$ and $\|c\|_\infty$; the smaller the norms are, the better the solution accuracy.

The interior-point method as implemented by **MOSEK** will converge toward optimality and primal and dual feasibility at the same rate [And09]. This means that if the optimizer is stopped prematurely then it is very unlikely that either the primal or dual solution is feasible. Another consequence is that in most cases all the tolerances, ε_p , ε_d , ε_g and ε_i , have to be relaxed together to achieve an effect.

In some cases the interior-point method terminates having found a solution not too far from meeting the optimality condition (13.11). A solution is defined as *near optimal* if scaling the termination tolerances ε_p , ε_d , ε_g and ε_i by the same factor $\varepsilon_n \in [1.0, +\infty]$ makes the condition (13.11) 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. Near infeasibility certificates are defined similarly. The value of ε_n can be adjusted with the parameter `dparam.intpnt_co_tol_near_rel`.

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

The Interior-point Log

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

```
Optimizer - threads          : 20
Optimizer - solved problem   : the primal
Optimizer - Constraints       : 1
```

Optimizer	-	Cones	:	2					
Optimizer	-	Scalar variables	:	6	conic	:	6		
Optimizer	-	Semi-definite variables:	0		scalarized	:	0		
Factor	-	setup time	:	0.00	dense det. time	:	0.00		
Factor	-	ML order time	:	0.00	GP order time	:	0.00		
Factor	-	nonzeros before factor	:	1	after factor	:	1		
Factor	-	dense dim.	:	0	flops	:	1.70e+01		
ITE	PFEAS	DFEAS	GFEAS	PRSTATUS	POBJ	DOBJ	MU	TIME	
0	1.0e+00	2.9e-01	3.4e+00	0.00e+00	2.414213562e+00	0.000000000e+00	1.0e+00	0.01	
1	2.7e-01	7.9e-02	2.2e+00	8.83e-01	6.969257574e-01	-9.685901771e-03	2.7e-01	0.01	
2	6.5e-02	1.9e-02	1.2e+00	1.16e+00	7.606090061e-01	6.046141322e-01	6.5e-02	0.01	
3	1.7e-03	5.0e-04	2.2e-01	1.12e+00	7.084385672e-01	7.045122560e-01	1.7e-03	0.01	
4	1.4e-08	4.2e-09	4.9e-08	1.00e+00	7.071067941e-01	7.071067599e-01	1.4e-08	0.01	

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

Using the same notation as in Section 13.4.1 the columns of the iteration log have the following meaning:

- **ITE:** Iteration index k .
- **PFEAS:** $\|Ax^k - b\tau^k\|_\infty$. The numbers in this column should converge monotonically towards zero but may stall at low level due to rounding errors.
- **DFEAS:** $\|A^T y^k + s^k - c\tau^k\|_\infty$. The numbers in this column should converge monotonically towards zero but may stall at low level due to rounding errors.
- **GFEAS:** $|-c^T x^k + b^T y^k - \kappa^k|$. The numbers in this column should converge monotonically towards zero but may stall at low level due to rounding errors.
- **PRSTATUS:** This number converges to 1 if the problem has an optimal solution whereas it converges to -1 if that is not the case.
- **POBJ:** $c^T x^k / \tau^k$. An estimate for the primal objective value.
- **DOBJ:** $b^T y^k / \tau^k$. An estimate for the dual objective value.
- **MU:** $\frac{(x^k)^T s^k + \tau^k \kappa^k}{n+1}$. The numbers in this column should always converge to zero.
- **TIME:** Time spent since the optimization started (in seconds).

13.5 Nonlinear Convex Optimization

13.5.1 The Interior-point Optimizer

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

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

For instance, the function

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

is differentiable everywhere whereas the function

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

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

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

$$x \geq 0$$

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

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

$$f(x) = \frac{1}{x} + \frac{1}{1-x}. \quad (13.12)$$

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, (13.12) can be reformulated as follows

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

Interior-point Termination Criteria

The parameters controlling when the general convex interior-point optimizer terminates are shown in Table 13.3.

Table 13.3: Parameters employed in termination criteria.

Parameter name	Purpose
<code>dparam.intpnt_nl_tol_pfeas</code>	Controls primal feasibility
<code>dparam.intpnt_nl_tol_dfeas</code>	Controls dual feasibility
<code>dparam.intpnt_nl_tol_rel_gap</code>	Controls relative gap
<code>dparam.intpnt_tol_infeas</code>	Controls when the problem is declared infeasible
<code>dparam.intpnt_nl_tol_mu_red</code>	Controls when the complementarity is reduced enough

THE OPTIMIZER FOR MIXED-INTEGER PROBLEMS

A problem is a mixed-integer optimization problem when one or more of the variables are constrained to be integer valued. Readers unfamiliar with integer optimization are recommended to consult some relevant literature, e.g. the book [Wol98] by Wolsey.

14.1 The Mixed-integer Optimizer Overview

MOSEK can solve mixed-integer

- linear,
- quadratic and quadratically constrained, and
- conic quadratic

problems, at least as long as they do not contain both quadratic objective or constraints and conic constraints at the same time. The mixed-integer optimizer is specialized for solving linear and conic optimization problems. Pure quadratic and quadratically constrained problems are automatically converted to conic form.

By default the mixed-integer optimizer is run-to-run deterministic. This means that if a problem is solved twice on the same computer with identical parameter settings and no time limit then the obtained solutions will be identical. If a time limit is set then this may not be case since the time taken to solve a problem is not deterministic. The mixed-integer optimizer is parallelized i.e. it can exploit multiple cores during the optimization.

The solution process can be split into these phases:

1. **Presolve:** See Section 13.1.
2. **Cut generation:** Valid inequalities (cuts) are added to improve the lower bound.
3. **Heuristic:** Using heuristics the optimizer tries to guess a good feasible solution. Heuristics can be controlled by the parameter `iparam.mio_heuristic_level`.
4. **Search:** The optimal solution is located by branching on integer variables.

14.2 Relaxations and bounds

It is important to understand that, in a worst-case scenario, the time required to solve integer optimization problems grows exponentially with the size of the problem (solving mixed-integer problems is NP-hard). For instance, a problem with n binary variables, may require time proportional to 2^n . The value of 2^n is huge even for moderate values of n .

In practice this implies that the focus should be on computing a near-optimal solution quickly rather than on locating an optimal solution. Even if the problem is only solved approximately, it is important to know how far the approximate solution is from an optimal one. In order to say something about the quality of an approximate solution the concept of *relaxation* is important.

Consider for example a mixed-integer optimization problem

$$\begin{aligned} z^* &= \text{minimize} && c^T x \\ &\text{subject to} && Ax = b, \\ &&& x \geq 0 \\ &&& x_j \in \mathbb{Z}, \quad \forall j \in \mathcal{J}. \end{aligned} \tag{14.1}$$

It has the continuous relaxation

$$\begin{aligned} \underline{z} &= \text{minimize} && c^T x \\ &\text{subject to} && Ax = b, \\ &&& x \geq 0 \end{aligned} \tag{14.2}$$

obtained simply by ignoring the integrality restrictions. The relaxation is a continuous problem, and therefore much faster to solve to optimality with a linear (or, in the general case, conic) optimizer. We call the optimal value \underline{z} the *objective bound*. The objective bound \underline{z} normally increases during the solution search process when the continuous relaxation is gradually refined.

Moreover, if \hat{x} is any feasible solution to (14.1) and

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

then

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

These two inequalities allow us to estimate the quality of the integer solution: it is no further away from the optimum than $\bar{z} - \underline{z}$ in terms of the objective value. Whenever a mixed-integer problem is solved **MOSEK** reports this lower bound so that the quality of the reported solution can be evaluated.

14.3 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. The issue of terminating the mixed-integer optimizer is rather delicate and the user has numerous possibilities of influencing it with various parameters. The mixed-integer optimizer employs a relaxed feasibility and optimality criterion to determine when a satisfactory solution is located.

A candidate solution that is feasible for the continuous relaxation is said to be an *integer feasible solution* if the criterion

$$\min(x_j - \lfloor x_j \rfloor, \lceil x_j \rceil - x_j) \leq \delta_1 \quad \forall j \in \mathcal{J}$$

is satisfied, meaning that x_j is at most δ_1 from the nearest integer.

Whenever the integer optimizer locates an integer feasible solution it will check if the criterion

$$\bar{z} - \underline{z} \leq \max(\delta_2, \delta_3 \max(10^{-10}, |\bar{z}|))$$

is satisfied. If this is the case, the integer optimizer terminates and reports the integer feasible solution as an optimal solution. If an optimal solution cannot be located after the time specified by the parameter `dparam.mio_disable_term_time` (in seconds), it may be advantageous to relax the termination criteria, and they become replaced with

$$\bar{z} - \underline{z} \leq \max(\delta_4, \delta_5 \max(10^{-10}, |\bar{z}|)).$$

Any solution satisfying those will now be reported as **near optimal** and the solver will be terminated (note that since this criterion depends on timing, the optimizer will not be run to run deterministic).

All the δ tolerances discussed above can be adjusted using suitable parameters — see [Table 14.1](#).

Table 14.1: Tolerances for the mixed-integer optimizer.

Tolerance	Parameter name
δ_1	<i>dparam.mio_tol_abs_relax_int</i>
δ_2	<i>dparam.mio_tol_abs_gap</i>
δ_3	<i>dparam.mio_tol_rel_gap</i>
δ_4	<i>dparam.mio_near_tol_abs_gap</i>
δ_5	<i>dparam.mio_near_tol_rel_gap</i>

In Table 14.2 some other common 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 14.2: Other parameters affecting the integer optimizer termination criterion.

Parameter name	De-layed	Explanation
<i>iparam.mio_max_num_branches</i>	Yes	Maximum number of branches allowed.
<i>iparam.mio_max_num_relaxs</i>	Yes	Maximum number of relaxations allowed.
<i>iparam.mio_max_num_solutions</i>	Yes	Maximum number of feasible integer solutions allowed.

14.4 Speeding Up the Solution Process

As mentioned previously, in many cases it is not possible to find an optimal solution to an integer optimization problem in a reasonable amount of time. Some suggestions to reduce the solution time are:

- Relax the termination criterion: In case the run time is not acceptable, the first thing to do is to relax the termination criterion — see Section 14.3 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].

14.5 Understanding Solution Quality

To determine the quality of the solution one should check the following:

- The problem status and solution status returned by **MOSEK**, as well as constraint violations in case of suboptimal solutions.
- The *optimality gap* defined as

$$\epsilon = |(\text{objective value of feasible solution}) - (\text{objective bound})| = |\bar{z} - \underline{z}|.$$

which measures how much the located solution can deviate from the optimal solution to the problem. The optimality gap can be retrieved through the information item *dinfitem.mio_obj_abs_gap*. Often it is more meaningful to look at the relative optimality gap normalized against the magnitude of the solution.

$$\epsilon_{\text{rel}} = \frac{|\bar{z} - \underline{z}|}{\max(10^{-10}, |\bar{z}|)}.$$

The relative optimality gap is available in *dinfitem.mio_obj_rel_gap*.

14.6 The Optimizer Log

Below is a typical log output from the mixed-integer optimizer:

```

Presolved problem: 6573 variables, 35728 constraints, 101258 non-zeros
Presolved problem: 0 general integer, 4294 binary, 2279 continuous
Clique table size: 1636
BRANCHES RELAXS  ACT_NDS  DEPTH    BEST_INT_OBJ      BEST_RELAX_OBJ      REL_GAP(%)  TIME
0          1        0        0        NA                1.8218819866e+07     NA           1.6
0          1        0        0        1.8331557950e+07   1.8218819866e+07     0.61         3.5
0          1        0        0        1.8300507546e+07   1.8218819866e+07     0.45         4.3
Cut generation started.
0          2        0        0        1.8300507546e+07   1.8218819866e+07     0.45         5.3
Cut generation terminated. Time = 1.43
0          3        0        0        1.8286893047e+07   1.8231580587e+07     0.30         7.5
15         18        1        0        1.8286893047e+07   1.8231580587e+07     0.30         10.5
31         34        1        0        1.8286893047e+07   1.8231580587e+07     0.30         11.1
51         54        1        0        1.8286893047e+07   1.8231580587e+07     0.30         11.6
91         94        1        0        1.8286893047e+07   1.8231580587e+07     0.30         12.4
171        174        1        0        1.8286893047e+07   1.8231580587e+07     0.30         14.3
331        334        1        0        1.8286893047e+07   1.8231580587e+07     0.30         17.9

[ ... ]

Objective of best integer solution : 1.825846762609e+07
Best objective bound                : 1.823311032986e+07
Construct solution objective         : Not employed
Construct solution # roundings       : 0
User objective cut value             : 0
Number of cuts generated             : 117
  Number of Gomory cuts              : 108
  Number of CMIR cuts                : 9
Number of branches                   : 4425
Number of relaxations solved         : 4410
Number of interior point iterations: 25
Number of simplex iterations         : 221131

```

The first lines contain a summary of the problem as seen by the optimizer. This is followed by the iteration log. The columns have the following meaning:

- **BRANCHES**: Number of branches generated.
- **RELAXS**: Number of relaxations solved.
- **ACT_NDS**: Number of active branch bound nodes.
- **DEPTH**: Depth of the recently solved node.
- **BEST_INT_OBJ**: The best integer objective value, \bar{z} .
- **BEST_RELAX_OBJ**: The best objective bound, \underline{z} .
- **REL_GAP(%)**: Relative optimality gap, $100\% \cdot \epsilon_{\text{rel}}$
- **TIME**: Time (in seconds) from the start of optimization.

Following that a summary of the optimization process is printed.

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

	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.

15.1 General Characteristics

The first part of the survey consists of a brief summary of the model's linear and quadratic constraints (indexed by i) and variables (indexed by j). The summary is divided into three subsections:

Constraints

- **upper bd** The number of upper bounded constraints, $\sum_{j=0}^{n-1} a_{ij}x_j \leq u_i^c$
- **lower bd** The number of lower bounded constraints, $l_i^c \leq \sum_{j=0}^{n-1} a_{ij}x_j$
- **ranged** The number of ranged constraints, $l_i^c \leq \sum_{j=0}^{n-1} a_{ij}x_j \leq u_i^c$
- **fixed** The number of fixed constraints, $l_i^c = \sum_{j=0}^{n-1} a_{ij}x_j = u_i^c$
- **free** The number of free constraints

Bounds

- **upper bd** The number of upper bounded variables, $x_j \leq u_j^x$
- **lower bd** The number of lower bounded variables, $l_k^x \leq x_j$
- **ranged** The number of ranged variables, $l_k^x \leq x_j \leq u_j^x$
- **fixed** The number of fixed variables, $l_k^x = x_j = u_j^x$
- **free** The number of free variables

Variables

- **cont** The number of continuous variables, $x_j \in \mathbb{R}$
- **bin** The number of binary variables, $x_j \in \{0, 1\}$
- **int** The number of general integer variables, $x_j \in \mathbb{Z}$

Only constraints, bounds and domains actually in the model will be reported on; if all entities in a section turn out to be of the same kind, the number will be replaced by **all** for brevity.

15.2 Objective

The second part of the survey focuses on (the linear part of) the objective, summarizing the optimization sense and the coefficients' absolute value range and distribution. The number of 0 (zero) coefficients is singled out (if any such variables are in the problem).

The range is displayed using three terms:

- **min |c|** The minimum absolute value among all coefficients
- **min |c|>0** The minimum absolute value among the nonzero coefficients
- **max |c|** The maximum absolute value among the coefficients

If some of these extrema turn out to be equal, the display is shortened accordingly:

- If **min |c|** is greater than zero, the **min |c|>0** term is obsolete and will not be displayed
- If only one or two different coefficients occur this will be displayed using **all** and an explicit listing of the coefficients

The absolute value distribution is displayed as a table summarizing the numbers by orders of magnitude (with a ratio of 10). Again, the number of variables with a coefficient of 0 (if any) is singled out. Each line of the table is headed by an interval (half-open intervals including their lower bounds), and is followed by the number of variables with their objective coefficient in this interval. Intervals with no elements are skipped.

15.3 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.

15.4 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).

15.5 Quadratic Constraints

The fifth part of the survey displays distributions for the nonzero elements in the gradient of the quadratic constraints, i.e. the nonzero row counts for the column vectors Qx . The table is similar to the tables for the linear constraints' nonzero row and column counts described in the survey's third part.

Note: Quadratic constraints may also have a linear part, but that will be included in the linear constraints survey; this means that if a problem has one or more pure quadratic constraints, part three of the survey will report an equal number of linear constraint rows with 0 (zero) nonzeros. Likewise, variables that appear in quadratic terms only will be reported as empty columns (0 nonzeros) in the linear constraint report.

15.6 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.

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.

16.1 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. 16.1.

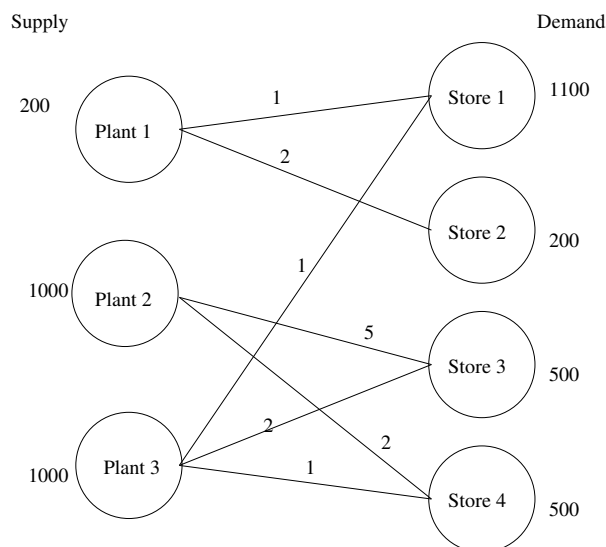


Fig. 16.1: Supply, demand and cost of transportation.

The problem represented in Fig. 16.1 is infeasible, since the total demand

$$2300 = 1100 + 200 + 500 + 500$$

exceeds the total supply

$$2200 = 200 + 1000 + 1000$$

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

$$\begin{array}{llllllllll}
 \text{minimize} & x_{11} & + & 2x_{12} & + & 5x_{23} & + & 2x_{24} & + & x_{31} & + & 2x_{33} & + & x_{34} \\
 \text{subject to} & x_{11} & + & x_{12} & & & & & & & & & & \leq 200, \\
 & & & & & x_{23} & + & x_{24} & & & & & & \leq 1000, \\
 & & & & & & & & x_{31} & + & x_{33} & + & x_{34} & \leq 1000, \\
 & x_{11} & & & & & & & + & x_{31} & & & & = 1100, \\
 & & x_{12} & & & & & & & & & & & = 200, \\
 & & & & x_{23} & + & & & & & x_{33} & & & = 500, \\
 & & & & & & x_{24} & + & & & & x_{34} & = 500, \\
 & x_{ij} & \geq 0.
 \end{array}
 \tag{16.1}$$

Solving problem (16.1) 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.

16.2 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 16.4) may assist you in finding the constraints causing the infeasibility.

Possible ways of relaxing your problem include:

- Increasing (decreasing) upper (lower) bounds on variables and constraints.
- Removing suspected constraints from the problem.

Returning to the transportation example, we discover that removing the fifth constraint

$$x_{12} = 200$$

makes the problem feasible.

16.3 Locating the Cause of Dual Infeasibility

A problem may also be *dual infeasible*. In this case the primal problem is often unbounded, meaning that feasible solutions exist such that the objective tends towards infinity. An example of a dual infeasible and primal unbounded problem is:

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

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.

16.3.1 A cautionary note

The problem

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

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.

16.4 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.

16.4.1 Example: Primal Infeasibility

We will keep working with the problem (16.1) written in LP format:

Listing 16.1: The code for problem (16.1).

```
\
\ 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
```

```

d1: + x11 + x31      = 1100
d2: + x12            = 200
d3: + x23 + x33      = 500
d4: + x24 + x34      = 500
bounds
end

```

16.4.2 Example: Dual Infeasibility

The following problem is dual to (16.1) and therefore it is dual infeasible.

Listing 16.2: The dual of problem (16.1).

```

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

$$(y_1, \dots, y_7) = (-1, 0, -1, 1, 1, 0, 0)$$

is a certificate of dual infeasibility (see Section 12.1.2.2) 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  2.000000e+02    NONE             2.000000e+00
6       x34       -1.000000e+00  1.100000e+03    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

```

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 12.1.2 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.

16.5 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

$$\begin{array}{ll} \text{minimize} & c^T x + c^f \\ \text{subject to} & l^c \leq Ax \leq u^c, \\ & l^x \leq x \leq u^x \end{array} \quad (16.2)$$

where the corresponding dual problem is

$$\begin{array}{ll} \text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c \\ & + (l^x)^T s_l^x - (u^x)^T s_u^x + c^f \\ \text{subject to} & A^T y + s_l^c - s_u^c = c, \\ & -y + s_l^x - s_u^x = 0, \\ & s_l^c, s_u^c, s_l^x, s_u^x \leq 0. \end{array} \quad (16.3)$$

We use the convention that for any bound that is not finite, the corresponding dual variable is fixed at zero (and thus will have no influence on the dual problem). For example

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

16.6 The Certificate of Primal Infeasibility

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

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

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

$$\begin{aligned} A^T y + s_l^{x*} - s_u^{x*} &= 0, \\ -y + s_l^{c*} - s_u^{c*} &= 0, \\ s_l^{c*}, s_u^{c*}, s_l^{x*}, s_u^{x*} &\leq 0. \end{aligned}$$

The well-known *Farkas Lemma* tells us that (16.2) 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,

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

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

16.7 The certificate of dual infeasibility

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

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

with negative objective value, where we use the definitions

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

and

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

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

$$\begin{aligned} c^T x^* &< 0, \\ \bar{l}^c &\leq Ax^* \leq \bar{u}^c, \\ \bar{l}^x &\leq x^* \leq \bar{u}^x \end{aligned} \tag{16.4}$$

The well-known Farkas Lemma tells us that (16.3) 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.

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.

17.1 Sensitivity Analysis for Linear Problems

17.1.1 The Optimal Objective Value Function

Assume that we are given the problem

$$\begin{aligned} z(l^c, u^c, l^x, u^x, c) = & \text{minimize} && c^T x \\ & \text{subject to} && l^c \leq Ax \leq u^c, \\ & && l^x \leq x \leq u^x, \end{aligned} \quad (17.1)$$

and we want to know how the optimal objective value changes as l_i^c is perturbed. To answer this question we define the perturbed problem for l_i^c as follows

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

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

$$f_{l_i^c}(\beta) \quad (17.2)$$

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 (17.2) shows the optimal objective value as a function of varying l_i^c with the other bounds fixed.

It is possible to prove that the function (17.2) is a piecewise linear and convex function, i.e. its graph may look like in Fig. 17.1 and Fig. 17.2.

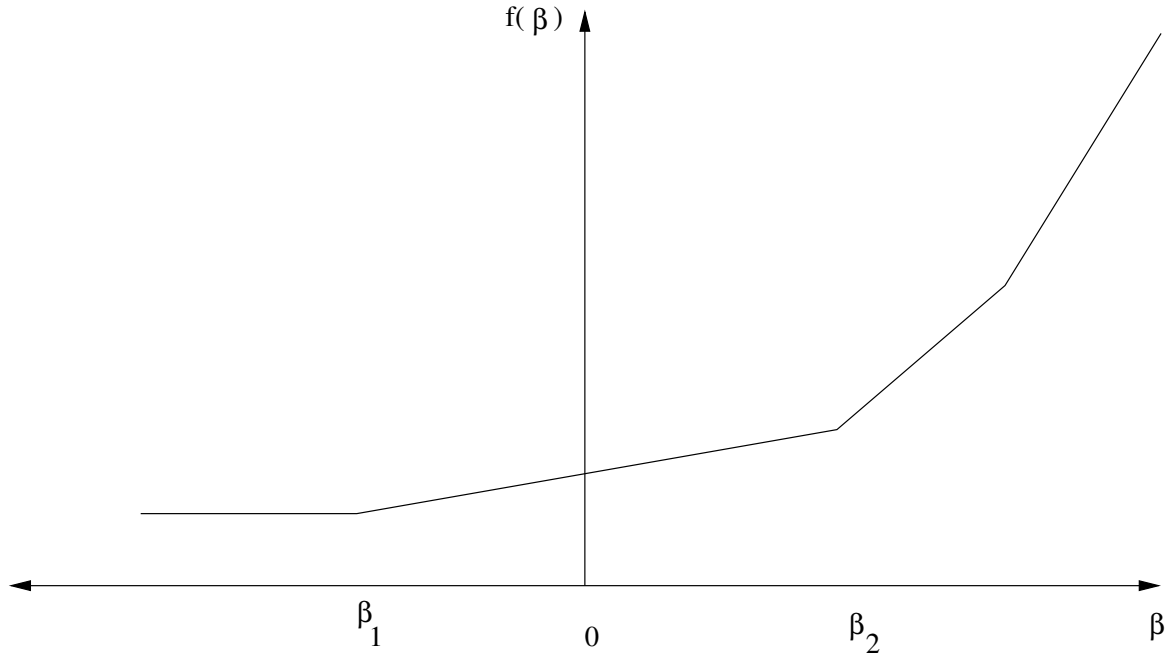


Fig. 17.1: $\beta = 0$ is in the interior of linearity interval.

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

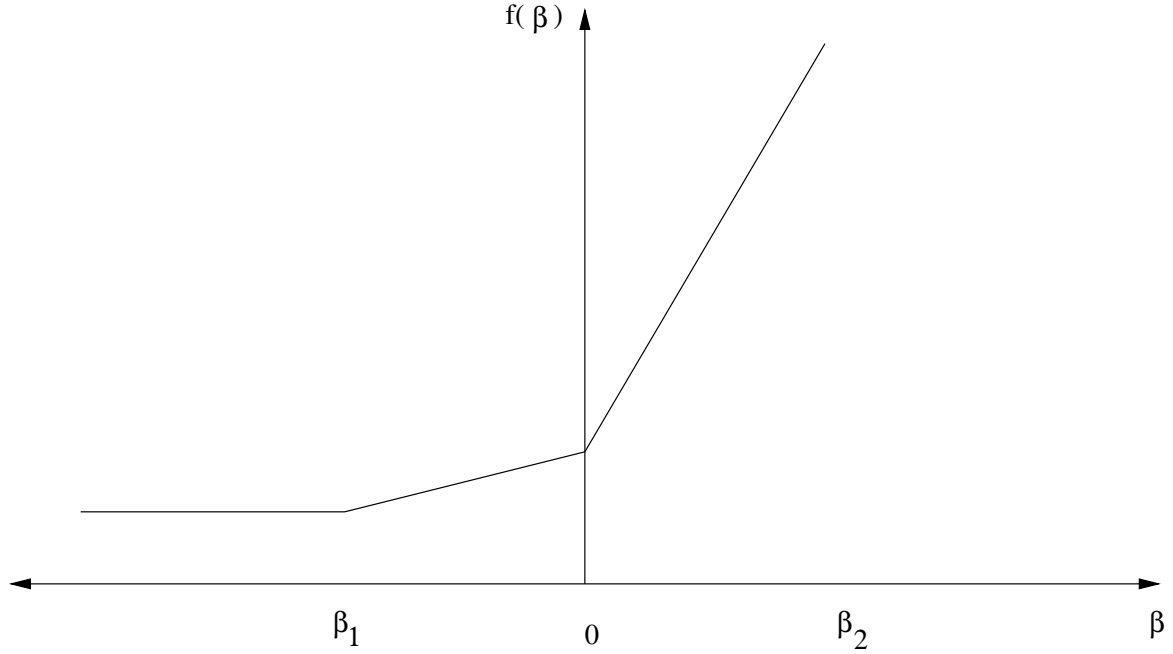
$$f'_{l_i^c}(\beta) = f'_{l_i^c}(0).$$

Since $f_{l_i^c}$ is not a smooth function $f'_{l_i^c}$ may not be defined at 0, as illustrated in Fig. 17.2. 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 (17.1) as well:

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

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

Fig. 17.2: $\beta = 0$ 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.

17.1.2 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.

17.1.3 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 (17.1), 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:

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

and

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

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

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

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

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

and

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

The linearity intervals and shadow prices for u_i^c , l_j^x , and u_j^x are computed similarly to l_i^c .

The left and right shadow prices for c_j denoted σ_1 and σ_2 respectively are computed as follows:

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

and

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

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

$$x_j^* \in [\sigma_1, \sigma_2].$$

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

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

and

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

17.1.4 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. 17.3.

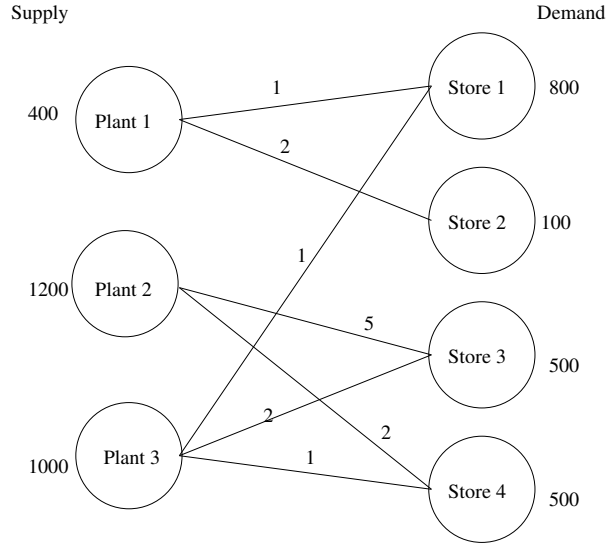


Fig. 17.3: Supply, demand and cost of transportation.

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

$$\begin{aligned} x_{11} + x_{12} & \leq 400, \\ x_{23} + x_{24} & \leq 1200, \\ x_{31} + x_{33} + x_{34} & \leq 1000, \\ x_{11} + x_{31} & = 800, \\ x_{12} + x_{32} & = 100, \\ x_{23} + x_{33} & = 500, \\ x_{24} + x_{34} & = 500, \\ x_{11}, x_{12}, x_{23}, x_{24}, x_{31}, x_{33}, x_{34} & \geq 0. \end{aligned} \tag{17.3}$$

The sensitivity parameters are shown in Table 17.1 and Table 17.2 for the basis type analysis and in Table 17.3 and Table 17.4 for the optimal partition type analysis.

Table 17.1: 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 17.2: 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

Table 17.3: 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 17.4: 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

$$\sigma_1 \beta = 3\beta.$$

17.2 Sensitivity Analysis with MOSEK

MOSEK provides the functions `Task.primalsensitivity` and `Task.dualsensitivity` for performing sensitivity analysis. The code in Listing 17.1 gives an example of its use.

Listing 17.1: Example of sensitivity analysis with the MOSEK Optimizer API for Java.

```

package com.mosek.example;
import mosek.*;

public class sensitivity {
    public static void main (String[] args) {
        // Since the value infinity is never used, we define
        // 'infinity' symbolic purposes only
        double
        infinity = 0;

        try (Env env = new Env();
            Task task = new Task(env, 0, 0)) {
            mosek.boundkey[] bkc = {
                mosek.boundkey.up, mosek.boundkey.up,
                mosek.boundkey.up, mosek.boundkey.fx,
                mosek.boundkey.fx, mosek.boundkey.fx,
                mosek.boundkey.fx
            };
            mosek.boundkey[] bkx = {
                mosek.boundkey.lo, mosek.boundkey.lo,
                mosek.boundkey.lo, mosek.boundkey.lo,
                mosek.boundkey.lo, mosek.boundkey.lo,
                mosek.boundkey.lo
            };
            int[] ptrb = {0, 2, 4, 6, 8, 10, 12};
            int[] ptre = {2, 4, 6, 8, 10, 12, 14};
            int[] sub = {0, 3, 0, 4, 1, 5, 1, 6, 2, 3, 2, 5, 2, 6};
            double[] blc = { -infinity, -infinity,
                            -infinity, 800, 100, 500, 500
                            };
            double[] buc = {400, 1200, 1000, 800, 100, 500, 500};
            double[] c = {1.0, 2.0, 5.0, 2.0, 1.0, 2.0, 1.0};
            double[] blx = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
            double[] bux = {infinity, infinity,
                            infinity, infinity,
                            infinity, infinity,
                            infinity
                            };
            double[] 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
                            };

            int numcon = 7; /* Number of constraints. */
            int numvar = 7; /* Number of variables. */
            int NUMANZ = 14; /* Number of non-zeros in A. */

            // Directs the log task stream to the user specified
            // method task_msg_obj.print
            task.set_Stream(
                mosek.streamtype.log,
                new mosek.Stream()
            { public void stream(String msg) { System.out.print(msg); } });

            task.inputdata(numcon, numvar,
                c,
                0.0,
                ptrb,
                ptre,
                sub,
                val,

```

```

        bkc,
        blc,
        buc,
        bkc,
        blx,
        bux);

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

    task.optimize();

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

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

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

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

    System.out.println("Results from sensitivity analysis on bounds:\n");

    System.out.println("For constraints:\n");
    for (int i = 0; i < 2; ++i)
        System.out.print("leftprice = " + leftpricei[i] +
                          " rightprice = " + rightpricei[i] +
                          " leftrange = " + leftrangei[i] +
                          " rightrange = " + rightrangei[i] + "\n");

    System.out.print("For variables:\n");
    for (int i = 0; i < 2; ++i)
        System.out.print("leftprice = " + leftpricej[i] +
                          " rightprice = " + rightpricej[i] +
                          " leftrange = " + leftrangej[i] +
                          " rightrange = " + rightrangej[i] + "\n");

    double[] leftprice = new double[2];
    double[] rightprice = new double[2];
    double[] leftrange = new double[2];

```

```
double[] rightrange = new double[2];
int subc[] = {2, 5};

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

System.out.println(
    "Results from sensitivity analysis on objective coefficients:"
);

for (int i = 0; i < 2; ++i)
    System.out.print("leftprice = " + leftprice[i] +
                     " rightprice = " + rightprice[i] +
                     " leftrange = " + leftrange[i] +
                     " rightrange = " + rightrange[i] + "\n");

} catch (mosek.Exception e)
    /* Catch both mosek.Error and mosek.Warning */
{
    System.out.println ("An error or warning was encountered");
    System.out.println (e.getMessage ());
    throw e;
}
}
```


API REFERENCE

This section contains the complete reference of the **MOSEK** Optimizer API for Java. It is organized as follows:

- *General API conventions.*
- **Methods:**
 - *Class Env* (The **MOSEK** environment)
 - *Class Task* (An optimization task)
 - *Browse by topic*
- **Optimizer parameters:**
 - *Double, Integer, String*
 - *Full list*
 - *Browse by topic*
- **Optimizer information items:**
 - *Double, Integer, Long*
- *Optimizer response codes*
- *Enumerations*
- *Exceptions*
- *User-defined class types*
- *Nonlinear API (SCopt)*

18.1 API Conventions

18.1.1 Function arguments

Naming Convention

In the definition of the **MOSEK** Optimizer API for Java 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 18.1 the variable names used to specify the problem parameters are listed.

Table 18.1: Naming conventions used in the **MOSEK** Optimizer API for Java.

API name	API type	Dimension	Related problem parameter
numcon	int		m
numvar	int		n
numcone	int		t
numqonz	int		q_{ij}^o
qosubi	int[]	numqonz	q_{ij}^o
qosubj	int[]	numqonz	q_{ij}^o
qoval	double[]	numqonz	q_{ij}^o
c	double[]	numvar	c_j
cfix	double		c^f
numqcnz	int		q_{ij}^k
qcsubk	int[]	qcnz	q_{ij}^k
qcsubi	int[]	qcnz	q_{ij}^k
qcsbj	int[]	qcnz	q_{ij}^k
qcval	double[]	qcnz	q_{ij}^k
aptrb	int[]	numvar	a_{ij}
aptre	int[]	numvar	a_{ij}
asub	int[]	aptre[numvar-1]	a_{ij}
aval	double[]	aptre[numvar-1]	a_{ij}
bkc	int[]	numcon	l_k^c and u_k^c
blc	double[]	numcon	l_k^c
buc	double[]	numcon	u_k^c
bkx	int[]	numvar	l_k^x and u_k^x
blx	double[]	numvar	l_k^x
bux	double[]	numvar	u_k^x

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

- The quadratic terms in the objective: $q_{qosubi[t],qosubj[t]}^o = qoval[t]$, $t = 0, \dots, numqonz - 1$.
- The linear terms in the objective : $c_j = c[j]$, $j = 0, \dots, numvar - 1$
- The fixed term in the objective : $c^f = cfix$.
- The quadratic terms in the constraints: $q_{qcsubi[t],qcsbj[t]}^{qcsbk[t]} = qcval[t]$, $t = 0, \dots, numqcnz - 1$
- The linear terms in the constraints: $a_{asub[t],j} = aval[t]$, $t = ptrb[j], \dots, ptre[j] - 1$, $j = 0, \dots, numvar - 1$

Passing arguments by reference

An argument described as **T** *by reference* indicates that the function interprets its given argument as a reference to a variable of type **T**. This usually means that the argument is used to output or update a value of type **T**. For example, suppose we have a function documented as

```
void foo (... , int[] nzc, ...)
```

- **nzc** (**int** *by reference*) – The number of nonzero elements in the matrix. (output)

Then it could be called as follows.

```
int nzc = new int[1];
foo (... , nzc, ...)
System.out.println("The number of nonzero elements: ", nzc[0])
```

Information about input/output arguments

The following are purely informational tags which indicate how **MOSEK** treats a specific function argument.

- (input) An input argument. It is used to input data to **MOSEK**.
- (output) An output argument. It can be a user-preallocated data structure, a reference, a string buffer etc. where **MOSEK** will output some data.
- (input/output) An input/output argument. **MOSEK** will read the data and overwrite it with new/updated information.

18.1.2 Bounds

The bounds on the constraints and variables are specified using the variables **bkc**, **bkc**, and **buc**. The components of the integer array **bkc** specify the bound type according to [Table 18.2](#)

Table 18.2: Symbolic key for variable and constraint bounds.

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

For instance **bkc[2]=*boundkey.lo*** means that $-\infty < l_2^c$ and $u_2^c = \infty$. Even if a variable or constraint is bounded only from below, e.g. $x \geq 0$, both bounds are inputted or extracted; the irrelevant value is ignored.

Finally, the numerical values of the bounds are given by

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

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

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

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

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

18.1.3 Vector Formats

Three different vector formats are used in the **MOSEK** API:

Full (dense) vector

This is simply an array where the first element corresponds to the first item, the second element to the second item etc. For example to get the linear coefficients of the objective in **task** with **numvar** variables, one would write

```
double[] c = new double[numvar];
task.getc(c);
```

Vector slice

A vector slice is a range of values from **first** up to and **not including last** entry in the vector, i.e. for the set of indices i such that $\text{first} \leq i < \text{last}$. For example, to get the bounds associated with constraints 2 through 9 (both inclusive) one would write

```
double[] upper_bound    = new double[8];
double[] lower_bound    = new double[8];
mosek.boundkey bound_key[]
                        = new mosek.boundkey[8];
task.getboundslice(mosek.accmode.con, 2,10,
                  bound_key,lower_bound,upper_bound);
```

Sparse vector

A sparse vector is given as an array of indexes and an array of values. The indexes need not be ordered. For example, to input a set of bounds associated with constraints number 1, 6, 3, and 9, one might write

```
int[] bound_index      = { 1, 6, 3, 9 };
mosek.boundkey[] bound_key
                      = { mosek.boundkey.fr,
                          mosek.boundkey.lo,
                          mosek.boundkey.up,
                          mosek.boundkey.fx };
double[] lower_bound = { 0.0, -10.0, 0.0, 5.0 };
double[] upper_bound = { 0.0, 0.0, 6.0, 5.0 };
task.putboundlist(mosek.accmode.con, bound_index,
                  bound_key,lower_bound,upper_bound);
```

18.1.4 Matrix Formats

The coefficient matrices in a problem are inputted and extracted in a sparse format. That means only the nonzero entries are listed.

Unordered Triplets

In unordered triplet format each entry is defined as a row index, a column index and a coefficient. For example, to input the A matrix coefficients for $a_{1,2} = 1.1$, $a_{3,3} = 4.3$, and $a_{5,4} = 0.2$, one would write as follows:

```
int[] subi = { 1, 3, 5 };
int[] subj = { 2, 3, 4 };
double[] cof = { 1.1, 4.3, 0.2 };
task.putaijlist(subi,subj,cof);
```

Please note that in some cases (like *Task.putaijlist*) *only* the specified indexes are modified — all other are unchanged. In other cases (such as *Task.putqconk*) the triplet format is used to modify *all* entries — entries that are not specified are set to 0.

Column or Row Ordered Sparse Matrix

In a sparse matrix format only the non-zero entries of the matrix are stored. **MOSEK** uses a sparse packed matrix format ordered either by columns or rows. Here we describe the column-wise format. The row-wise format is based on the same principle.

Column ordered sparse format

A sparse matrix in column ordered format is essentially a list of all non-zero entries read column by column from left to right and from top to bottom within each column. The exact representation uses four arrays:

- **asub**: Array of size equal to the number of nonzeros. List of row indexes.
- **aval**: Array of size equal to the number of nonzeros. List of non-zero entries of A ordered by columns.
- **ptrb**: Array of size `numcol`, where `ptrb[j]` is the position of the first value/index in **aval**/**asub** for the j -th column.
- **ptre**: Array of size `numcol`, where `ptre[j]` is the position of the last value/index plus one in **aval**/**asub** for the j -th column.

With this representation the values of a matrix A with `numcol` columns are assigned using:

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

As an example consider the matrix

$$A = \begin{bmatrix} 1.1 & & 1.3 & 1.4 & \\ & 2.2 & & & 2.5 \\ 3.1 & & & 3.4 & \\ & & 4.4 & & \end{bmatrix} \quad (18.1)$$

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

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

Fig. 18.1 illustrates how the matrix A in (18.1) is represented in column ordered sparse matrix format.

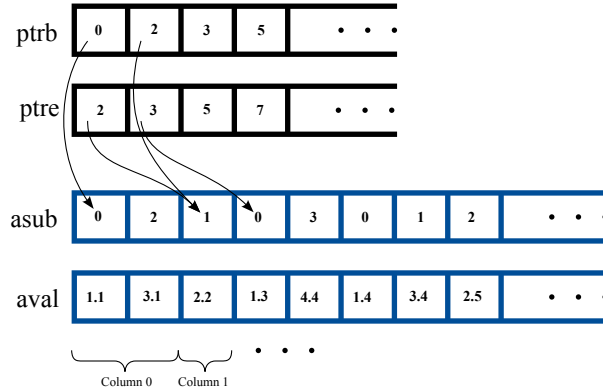


Fig. 18.1: The matrix A (18.1) represented in column ordered packed sparse matrix format.

Column ordered sparse format with nonzeros

Note that `nzc[j] := pre[j] - ptrb[j]` is exactly the number of nonzero elements in the j -th column of A . In some functions a sparse matrix will be represented using the equivalent dataset **asub**, **aval**, **ptrb**, **nzc**. The matrix A (18.1) would now be represented as:

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

Row ordered sparse matrix

The matrix A (18.1) can also be represented in the row ordered sparse matrix format as:

$$\begin{aligned}
\text{ptrb} &= [0, 3, 5, 7], \\
\text{ptre} &= [3, 5, 7, 8], \\
\text{asub} &= [0, 2, 3, 1, 4, 0, 3, 2], \\
\text{aval} &= [1.1, 1.3, 1.4, 2.2, 2.5, 3.1, 3.4, 4.4].
\end{aligned}$$

18.2 Functions grouped by topic

Basis matrix

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

Bound data

- *Task.putbound* – Changes the bound for either one constraint or one variable.
- *Task.putboundslice* – Changes the bounds for a slice of constraints or variables.
- *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.putvarbound* – Changes the bound for one variable.
- *Task.putvarboundlist* – Changes the bounds of a list of variables.
- Infrequent: *Task.chgconbound*, *Task.chgvarbound*, *Task.getbound*, *Task.getboundslice*, *Task.getconbound*, *Task.getconboundslice*, *Task.getvarbound*, *Task.getvarboundslice*, *Task.putboundlist*
- Deprecated: ~~*Task.chgbound*~~

Conic constraint data

- *Task.appendcone* – Appends a new conic constraint to the problem.
- *Task.putcone* – Replaces a conic constraint.
- *Task.removecones* – Removes a number of conic constraints from the problem.
- Infrequent: *Task.appendconeseq*, *Task.appendconesseq*, *Task.getcone*, *Task.getconeinfo*, *Task.getnumcone*, *Task.getnumconemem*

Data file

- *Task.readsolution* – Reads a solution from a file.
- *Task.writedata* – Writes problem data to a file.
- *Task.writesolution* – Write a solution to a file.
- Infrequent: *Task.readdata*, *Task.readdataformat*, *Task.readparamfile*, *Task.writejsonsol*, *Task.writeparamfile*

Environment management

- *Env.licensecleanup* – Stops all threads and delete all handles used by the license system.
- *Env.putlicensedebug* – Enables debug information for the license system.
- *Env.putlicensepath* – Set the path to the license file.
- *Env.putlicensewait* – Control whether mosek should wait for an available license if no license is available.
- Infrequent: *Env.checkinall*, *Env.checkinlicense*, *Env.checkoutlicense*, *Env.putlicensecode*

Infeasibility diagnostics

- *Task.getinfeasiblesubproblem* – Obtains an infeasible subproblem.
- *Task.primalrepair* – Repairs a primal infeasible optimization problem by adjusting the bounds on the constraints and variables.

Linear algebra

- *Env.axy* – Computes vector addition and multiplication by a scalar.
- *Env.computesparseseholesky* – Computes a Cholesky factorization of sparse matrix.
- *Env.dot* – Computes the inner product of two vectors.
- *Env.gemm* – Performs a dense matrix multiplication.
- *Env.gemv* – Computes dense matrix times a dense vector product.
- *Env.potrf* – Computes a Cholesky factorization of a dense matrix.
- *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 eigenvalues and eigenvectors of a symmetric dense matrix, and thus its eigenvalue decomposition.
- *Env.syrk* – Performs a rank-k update of a symmetric matrix.

Linear constraint data

- *Task.appendcons* – Appends a number of constraints to the optimization task.
- *Task.getnumcon* – Obtains the number of constraints.
- *Task.putconboundslice* – Changes the bounds for a slice of the constraints.
- *Task.removecons* – Removes a number of constraints.
- Infrequent: *Task.getmaxnumcon*

Logging

- *Task.linkfiletoostream* – Directs all output from a task stream to a file.
- Infrequent: *Env.linkfiletoostream*

Memory

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

Naming

- *Task.putbarvarname* – Sets the name of a semidefinite variable.
- *Task.putconename* – Sets the name of a cone.
- *Task.putconname* – Sets the name of a constraint.
- *Task.putobjname* – Assigns a new name to the objective.
- *Task.puttaskname* – Assigns a new name to the task.
- *Task.putvarname* – Sets the name of a variable.
- Infrequent: *Task.getbarvarname*, *Task.getbarvarnameindex*, *Task.getbarvarnamelen*, *Task.getconename*, *Task.getconenameindex*, *Task.getconenamelen*, *Task.getconname*, *Task.getconnameindex*, *Task.getconnamelen*, *Task.getobjname*, *Task.getobjnamelen*, *Task.gettaskname*, *Task.gettasknamelen*, *Task.getvarname*, *Task.getvarnameindex*, *Task.getvarnamelen*

Objective data

- *Task.putcfix* – Replaces the fixed term in the objective.
- *Task.putobjsense* – Sets the objective sense.
- Infrequent: *Task.getobjsense*

Optimization

- *Task.optimize* – Optimizes the problem.

Optimizer statistics

- *Task.getdoublinf* – Obtains a double information item.
- *Task.getintinf* – Obtains an integer information item.
- *Task.getlintinf* – Obtains a long integer information item.
- Infrequent: *Task.getinfindex*, *Task.getinfmax*, *Task.getinfname*

Parameter management

- Infrequent: *Task.getnumparam*, *Task.getparammax*, *Task.getparamname*, *Task.isdouparname*, *Task.isintparname*, *Task.isstrparname*, *Task.setdefaults*

Parameters (get)

- Infrequent: *Task.getdoupparam*, *Task.getintparam*, *Task.getstrparam*, *Task.getstrparamlen*

Parameters (put)

- *Task.putdouparam* – Sets a double parameter.
- *Task.putintparam* – Sets an integer parameter.
- *Task.putstrparam* – Sets a string parameter.
- Infrequent: *Task.putnadouparam*, *Task.putnaintparam*, *Task.putnastrparam*, *Task.putparam*

Scalar variable data

- *Task.appendvars* – Appends a number of variables to the optimization task.
- *Task.getnumvar* – Obtains the number of variables.
- *Task.putacol* – Replaces all elements in one column of the linear constraint matrix.
- *Task.putaij* – Changes a single value in the linear coefficient matrix.
- *Task.putarow* – Replaces all elements in one row of the linear constraint matrix.
- *Task.putcj* – Modifies one linear coefficient in the objective.
- *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.putvarboundslice* – Changes the bounds for a slice of the variables.
- *Task.putvartype* – Sets the variable type of one variable.
- *Task.removevars* – Removes a number of variables.
- Infrequent: *Task.commitchanges*, *Task.getacol*, *Task.getacolnumnz*, *Task.getacolslicetrip*, *Task.getaij*, *Task.getarow*, *Task.getarownumnz*, *Task.getarowslicetrip*, *Task.getaslice*, *Task.getc*, *Task.getcfix*, *Task.getcj*, *Task.getcslice*, *Task.getlenbarvarj*, *Task.getmaxnumanz*, *Task.getmaxnumqnz*, *Task.getmaxnumvar*, *Task.getnumanz*, *Task.getnumanz64*, *Task.getnumintvar*, *Task.getnumqconknz*, *Task.getnumqobjnz*, *Task.getnumsymmat*, *Task.getqconk*, *Task.getqobj*, *Task.getqobjij*, *Task.getsparsesymmat*, *Task.getsymmatinfo*, *Task.getvartype*, *Task.getvartypelist*, *Task.putacollist*, *Task.putacolslice*, *Task.putaijlist*, *Task.putarowlist*, *Task.putarowslice*, *Task.putclist*, *Task.putcslice*, *Task.putmaxnumanz*, *Task.putmaxnumqnz*, *Task.putmaxnumvar*, *Task.putvartypelist*

Sensitivity analysis

- *Task.dualsensitivity* – Performs sensitivity analysis on objective coefficients.
- *Task.primalsensitivity* – Perform sensitivity analysis on bounds.
- *Task.sensitivityreport* – Creates a sensitivity report.

Solution (get)

- *Task.getbarsj* – Obtains the dual solution for a semidefinite variable.
- *Task.getbarxj* – Obtains the primal solution for a semidefinite variable.
- *Task.getskcslice* – Obtains the status keys for a slice of the constraints.

- *Task.getskxslice* – Obtains the status keys for a slice of the scalar variables.
- *Task.getslcslice* – Obtains a slice of the slc vector for a solution.
- *Task.getslxslice* – Obtains a slice of the slx vector for a solution.
- *Task.getsnxslice* – Obtains a slice of the snx vector for a solution.
- *Task.getsucslice* – Obtains a slice of the suc vector for a solution.
- *Task.getsuxslice* – Obtains a slice of the sux vector for a solution.
- *Task.getxcslice* – Obtains a slice of the xc vector for a solution.
- *Task.getxxslice* – Obtains a slice of the xx vector for a solution.
- *Task.getyslice* – Obtains a slice of the y vector for a solution.
- Infrequent: *Task.getreducedcosts*, *Task.getskc*, *Task.getskx*, *Task.getslc*, *Task.getslx*, *Task.getsnx*, *Task.getsolution*, *Task.getsolutioni*, *Task.getsolutionslice*, *Task.getsuc*, *Task.getsux*, *Task.getxc*, *Task.getxx*, *Task.gety*

Solution (put)

- *Task.putbarsj* – Sets the dual solution for a semidefinite variable.
- *Task.putbarxj* – Sets the primal solution for a semidefinite variable.
- *Task.putskcslice* – Sets the status keys for a slice of the constraints.
- *Task.putskxslice* – Sets the status keys for a slice of the variables.
- *Task.putslcslice* – Sets a slice of the slc vector for a solution.
- *Task.putslxslice* – Sets a slice of the slx 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.putsucslice* – Sets a slice of the suc vector for a solution.
- *Task.putsuxslice* – Sets a slice of the sux vector for a solution.
- *Task.putxcslice* – Sets a slice of the xc vector for a solution.
- *Task.putxxslice* – Obtains a slice of the xx vector for a solution.
- *Task.putyslice* – Sets a slice of the y vector for a solution.
- Infrequent: *Task.putskc*, *Task.putskx*, *Task.putslc*, *Task.putslx*, *Task.putsnx*, *Task.putsuc*, *Task.putsux*, *Task.putxc*, *Task.putxx*, *Task.puty*

Solution information

- *Task.getdualobj* – Computes the dual objective value associated with the solution.
- *Task.getdualsolutionnorms* – Compute norms of the dual solution.
- *Task.getdviolbarvar* – Computes the violation of dual solution for a set of semidefinite 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 scalar variables.

- *Task.getprimalobj* – Computes the primal objective value for the desired solution.
- *Task.getprimalsolutionnorms* – Compute norms of the primal solution.
- *Task.getprosta* – Obtains the problem status.
- *Task.getpviolbarvar* – Computes the violation of a primal solution for a list of semidefinite variables.
- *Task.getpviolcon* – Computes the violation of a primal solution associated to a constraint.
- *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 scalar variables.
- *Task.getsolsta* – Obtains the solution status.
- *Task.getsolutioninfo* – Obtains information about of a solution.
- *Task.solutiondef* – Checks whether a solution is defined.

Symmetric matrix variable data

- *Task.appendbarvars* – Appends semidefinite variables to the problem.
- *Task.appendsparsesymmat* – Appends a general sparse symmetric matrix to the storage of symmetric matrices.
- *Task.putbaraij* – Inputs an element of barA.
- *Task.putbarcj* – Changes one element in barc.
- Infrequent: *Task.getbarablocktriplet*, *Task.getbaraidx*, *Task.getbaraidxij*, *Task.getbaraidxinfo*, *Task.getbarasparsity*, *Task.getbarcblocktriplet*, *Task.getbarcidx*, *Task.getbarcidxinfo*, *Task.getbarcidxj*, *Task.getbarcsparsity*, *Task.getdimbarvarj*, *Task.getmaxnumbarvar*, *Task.getnumbarablocktriplets*, *Task.getnumbaranz*, *Task.getnumbarcblocktriplets*, *Task.getnumbarcnz*, *Task.getnumbarvar*, *Task.putbarablocktriplet*, *Task.putbarcblocktriplet*, *Task.putmaxnumbarvar*, *Task.removebarvars*

Task diagnostics

- *Task.checkconvexity* – Checks if a quadratic optimization problem is convex.
- *Task.getprobtype* – Obtains the problem type.
- *Task.onesolutionsummary* – Prints a short summary of a specified solution.
- *Task.optimizersummary* – Prints a short summary with optimizer statistics from last optimization.
- *Task.solutionsummary* – Prints a short summary of the current solutions.
- *Task.updatesolutioninfo* – Update the information items related to the solution.
- Infrequent: *Task.analyzenames*, *Task.analyzeproblem*, *Task.analyzesolution*, *Env.echointro*, *Task.readsummary*

Task management

- Infrequent: *Task.deletesolution*, *Env.getcodedesc*, *Task.getmaxnumcone*, *Task.inputdata*, *Task.putmaxnumcon*, *Task.putmaxnumcone*

Other

- *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.
- *Env.getversion* – Obtains MOSEK version information.
- *Task.optimizermt* – Offload the optimization task to a solver server.
- *Task.putsolutionyi* – Inputs the dual variable of a solution.
- *Task.readtask* – Load task data from a file.
- *Task.resizetask* – Resizes an optimization task.
- *Task.set_InfoCallback* – Receive callbacks with solver status and information during optimization.
- *Task.set_ItgSolutionCallback* – Receive callbacks with solution updates from the mixed-integer optimizer.
- *Task.set_Progress* – Receive callbacks about current status of the solver during optimization.
- *Task.set_Stream* – Directs all output from a task stream to a callback object.
- *Env.set_Stream* – Directs all output from an environment stream to a callback object.
- *Task.toconic* – In-place reformulation of a QCQP to a COP
- *Task.unset_Progress* – Deactivates all user callback functions.
- *Task.writetask* – Write a complete binary dump of the task data.
- Infrequent: *Task.getapienummz*, *Task.getaslicenummz*, *Task.strtoconetype*,
 Task.strtosk

18.3 Class Env

`mosek.Env`

The **MOSEK** global environment.

`Env.Env`

`Env()`

`Env(String dbgfile)`

Constructor of a new environment.

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

`Env.axpy`

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

Adds αx to y , i.e. performs the update

$$y := \alpha x + y.$$

Note that the result is stored overwriting y .

Parameters

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

Groups *Linear algebra*

`Env.checkinall`

```
void checkinall ()
```

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

Groups *Environment management*

`Env.checkinlicense`

```
void checkinlicense (mosek.feature feature)
```

Check in a license feature to the license server. By default all licenses consumed by functions using a single environment are kept checked out for the lifetime of the **MOSEK** environment. This function checks in a given license feature back to the license server immediately.

If the given license feature is not checked out at all, or it is in use by a call to *Task.optimize*, calling this function has no effect.

Please note that returning a license to the license server incurs a small overhead, so frequent calls to this function should be avoided.

Parameters **feature** (*mosek.feature*) – Feature to check in to the license system. (input)

Groups *Environment management*

`Env.checkoutlicense`

```
void checkoutlicense (mosek.feature feature)
```

Checks out a license feature from the license server. Normally the required license features will be automatically checked out the first time they are needed by the function *Task.optimize*. This function can be used to check out one or more features ahead of time.

The feature will remain checked out until the environment is deleted or the function *Env.checkinlicense* is called.

If a given feature is already checked out when this function is called, the call has no effect.

Parameters **feature** (*mosek.feature*) – Feature to check out from the license system. (input)

Groups *Environment management*

`Env.computesparsedcholesky`

```

void computesparsecholesky
(int multithread,
 int ordermethod,
 double tolsingular,
 int[] anzc,
 long[] aptrc,
 int[] asubc,
 double[] avalc,
 int[][] perm,
 double[][] diag,
 int[][] lnzc,
 long[][] lptrc,
 long[] lensubnval,
 int[][] lsubc,
 double[][] lvalc)

```

The function computes a Cholesky factorization of a sparse positive semidefinite matrix. Sparsity is exploited during the computations to reduce the amount of space and work required. Both the input and output matrices are represented using the sparse format.

To be precise, given a symmetric matrix $A \in \mathbb{R}^{n \times n}$ the function computes a nonsingular lower triangular matrix L , a diagonal matrix D and a permutation matrix P such that

$$LL^T - D = PAP^T.$$

If `ordermethod` is zero then reordering heuristics are not employed and P is the identity.

If a pivot during the computation of the Cholesky factorization is less than

$$-\rho \cdot \max((PAP^T)_{jj}, 1.0)$$

then the matrix is declared negative semidefinite. On the hand if a pivot is smaller than

$$\rho \cdot \max((PAP^T)_{jj}, 1.0),$$

then D_{jj} is increased from zero to

$$\rho \cdot \max((PAP^T)_{jj}, 1.0).$$

Therefore, if A is sufficiently positive definite then D will be the zero matrix. Here ρ is set equal to value of `tolsingular`.

Parameters

- `multithread` (int) – If nonzero then the function may exploit multiple threads. (input)
- `ordermethod` (int) – If nonzero, then a sparsity preserving ordering will be employed. (input)
- `tolsingular` (double) – A positive parameter controlling when a pivot is declared zero. (input)
- `anzc` (int[]) – `anzc[j]` is the number of nonzeros in the j -th column of A . (input)
- `aptrc` (long[]) – `aptrc[j]` is a pointer to the first element in column j of A . (input)
- `asubc` (int[]) – Row indexes for each column stored in increasing order. (input)
- `avalc` (double[]) – The value corresponding to row indexed stored in `asubc`. (input)
- `perm` (int[] *by reference*) – Permutation array used to specify the permutation matrix P computed by the function. (output)

- `diag` (`double[]` *by reference*) – The diagonal elements of matrix D . (output)
- `lnzc` (`int[]` *by reference*) – `lnzc[j]` is the number of non zero elements in column j of L . (output)
- `lptrc` (`long[]` *by reference*) – `lptrc[j]` is a pointer to the first row index and value in column j of L . (output)
- `lensubnval` (`long` *by reference*) – Number of elements in `lsubc` and `lvalc`. (output)
- `lsubc` (`int[]` *by reference*) – Row indexes for each column stored in increasing order. (output)
- `lvalc` (`double[]` *by reference*) – The values corresponding to row indexed stored in `lsubc`. (output)

Groups *Linear algebra*

`Env.dot`

```
void dot
(int n,
 double[] x,
 double[] y,
 double[] xty)
```

Computes the inner product of two vectors x, y of length $n \geq 0$, i.e

$$x \cdot y = \sum_{i=1}^n x_i y_i.$$

Note that if $n = 0$, then the result of the operation is 0.

Parameters

- `n` (`int`) – Length of the vectors. (input)
- `x` (`double[]`) – The x vector. (input)
- `y` (`double[]`) – The y vector. (input)
- `xty` (`double` *by reference*) – The result of the inner product between x and y . (output)

Groups *Linear algebra*

`Env.echointro`

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

Prints an intro to message stream.

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

Groups *Task diagnostics*

`Env.gemm`

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

```
double alpha,
double[] a,
double[] b,
double beta,
double[] c)
```

Performs a matrix multiplication plus addition of dense matrices. Given A , B and C of compatible dimensions, this function computes

$$C := \alpha op(A)op(B) + \beta C$$

where α, β are two scalar values. The function $op(X)$ denotes X if `transX` is `transpose.no`, or X^T if set to `transpose.yes`. The matrix C has m rows and n columns, and the other matrices must have compatible dimensions.

The result of this operation is stored in C .

Parameters

- `transa` (`mosek.transpose`) – Indicates whether the matrix A must be transposed. (input)
- `transb` (`mosek.transpose`) – Indicates whether the matrix B must be transposed. (input)
- `m` (`int`) – Indicates the number of rows of matrix C . (input)
- `n` (`int`) – Indicates the number of columns of matrix C . (input)
- `k` (`int`) – Specifies the common dimension along which $op(A)$ and $op(B)$ are multiplied. For example, if neither A nor B are transposed, then this is the number of columns in A and also the number of rows in B . (input)
- `alpha` (`double`) – A scalar value multiplying the result of the matrix multiplication. (input)
- `a` (`double[]`) – The pointer to the array storing matrix A in a column-major format. (input)
- `b` (`double[]`) – The pointer to the array storing matrix B in a column-major format. (input)
- `beta` (`double`) – A scalar value that multiplies C . (input)
- `c` (`double[]`) – The pointer to the array storing matrix C in a column-major format. (input/output)

Groups *Linear algebra*

`Env.gemv`

```
void gemv
(mosek.transpose transa,
 int m,
 int n,
 double alpha,
 double[] a,
 double[] x,
 double beta,
 double[] y)
```

Computes the multiplication of a scaled dense matrix times a dense vector, plus a scaled dense vector. Precisely, if `trans` is `transpose.no` then the update is

$$y := \alpha Ax + \beta y,$$

and if `trans` is *transpose.yes* then

$$y := \alpha A^T x + \beta y,$$

where α, β are scalar values and A is a matrix with m rows and n columns.

Note that the result is stored overwriting y .

Parameters

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

Groups *Linear algebra*

`Env.getcodedesc`

```
static void getcodedesc
(mosek.rescode code,
 StringBuffer symname,
 StringBuffer str)
```

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

Parameters

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

Groups *Task management*

`Env.getversion`

```
static void getversion
(int[] major,
 int[] minor,
 int[] build,
 int[] revision)
```

Obtains **MOSEK** version information.

Parameters

- `major` (`int` *by reference*) – Major version number. (output)
- `minor` (`int` *by reference*) – Minor version number. (output)
- `build` (`int` *by reference*) – Build number. (output)
- `revision` (`int` *by reference*) – Revision number. (output)

Env.licensecleanup

```
static void licensecleanup ()
```

Stops all threads and deletes all handles used by the license system. If this function is called, it must be called as the last **MOSEK** API call. No other **MOSEK** API calls are valid after this.

Groups *Environment management*

Env.linkfiletostream

```
void linkfiletostream  
(mosek.streamtype whichstream,  
 String filename,  
 int append)
```

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

Parameters

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

Groups *Logging*

Env.potrf

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

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

Parameters

- `uplo` (*mosek.uplo*) – Indicates whether the upper or lower triangular part of the matrix is stored. (input)
- `n` (*int*) – Dimension of the symmetric matrix. (input)
- `a` (*double[]*) – A symmetric matrix stored in column-major order. Only the lower or the upper triangular part is used, accordingly with the `uplo` parameter. It will contain the result on exit. (input/output)

Groups *Linear algebra*

Env.putlicensecode

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

Input a runtime license code.

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

Groups *Environment management*

Env.putlicensedebug

```
void putlicensedebug (int licdebug)
```

Enables debug information for the license system. If `licdebug` is non-zero, then **MOSEK** will print debug info regarding the license checkout.

Parameters `licdebug (int)` – Whether license checkout debug info should be printed. (input)

Groups *Environment management*

`Env.putlicensepath`

```
void putlicensepath (String licensepath)
```

Set the path to the license file.

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

Groups *Environment management*

`Env.putlicensewait`

```
void putlicensewait (int licwait)
```

Control whether **MOSEK** should wait for an available license if no license is available. If `licwait` is non-zero, then **MOSEK** will wait for `licwait-1` milliseconds between each check for an available license.

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

Groups *Environment management*

`Env.set_Stream`

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

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

Can for example be called as:

```
env.set_Stream(mosek.streamtype.log, new Stream() { public void stream(String s) { System.
    ↪out.print(s); } } );
```

Parameters

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

`Env.sparsetriangularsolvedense`

```
void sparsetriangularsolvedense
(mosek.transpose transposed,
 int[] lnzc,
 long[] lptrc,
 int[] lsubc,
```

```
double[] lvalc,
double[] b)
```

The function solves a triangular system of the form

$$Lx = b$$

or

$$L^T x = b$$

where L is a sparse lower triangular nonsingular matrix. This implies in particular that diagonals in L are nonzero.

Parameters

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

Groups *Linear algebra*

Env.syeig

```
void syeig
(mosek.uplo uplo,
 int n,
 double[] a,
 double[] w)
```

Computes all eigenvalues of a real symmetric matrix A . Given a matrix $A \in \mathbb{R}^{n \times n}$ it returns a vector $w \in \mathbb{R}^n$ containing the eigenvalues of A .

Parameters

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

Groups *Linear algebra*

Env.syevd

```

void syevd
(mosek.uplo uplo,
 int n,
 double[] a,
 double[] w)

```

Computes all the eigenvalues and eigenvectors a real symmetric matrix. Given the input matrix $A \in \mathbb{R}^{n \times n}$, this function returns a vector $w \in \mathbb{R}^n$ containing the eigenvalues of A and it also computes the eigenvectors of A . Therefore, this function computes the eigenvalue decomposition of A as

$$A = UVU^T,$$

where $V = \text{diag}(w)$ and U contains the eigenvectors of A .

Note that the matrix U overwrites the input data A .

Parameters

- `uplo` (*mosek.uplo*) – Indicates whether the upper or lower triangular part is used. (input)
- `n` (`int`) – Dimension of the symmetric input matrix. (input)
- `a` (`double[]`) – A symmetric matrix A stored in column-major order. Only the part indicated by `uplo` is used. On exit it will be overwritten by the matrix U . (input/output)
- `w` (`double[]`) – Array of length at least `n` containing the eigenvalues of A . (output)

Groups *Linear algebra*

`Env.syrk`

```

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

```

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

Given a symmetric matrix $C \in \mathbb{R}^{n \times n}$, two scalars α, β and a matrix A of rank $k \leq n$, it computes either

$$C := \alpha AA^T + \beta C,$$

when `trans` is set to *transpose.no* and $A \in \mathbb{R}^{n \times k}$, or

$$C := \alpha A^T A + \beta C,$$

when `trans` is set to *transpose.yes* and $A \in \mathbb{R}^{k \times n}$.

Only the part of C indicated by `uplo` is used and only that part is updated with the result.

Parameters

- `uplo` (*mosek.uplo*) – Indicates whether the upper or lower triangular part of C is used. (input)

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

Groups *Linear algebra*

18.4 Class Task

`mosek.Task`

Represents an optimization task.

`Task.Task`

```
Task(mosek.Env env)
```

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

```
Task(mosek.Task task)
```

Constructor of a new optimization task.

Parameters

- `env` (*Env*) – Parent environment. (input)
- `numcon` (`int`) – Initial number of constraints in the task. (input)
- `numvar` (`int`) – Initial number of variables in the task. (input)
- `task` (*Task*) – A task that will be cloned. (input)

`Task.analyzenames`

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

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

Parameters

- `whichstream` (*mosek.streamtype*) – Index of the stream. (input)
- `nametype` (*mosek.nametype*) – The type of names e.g. valid in MPS or LP files. (input)

Groups *Task diagnostics***Task.analyzeproblem**

```
void analyzeproblem (mosek.streamtype whichstream)
```

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

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

Groups *Task diagnostics***Task.analyzesolution**

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

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

By default this function prints information about the largest infeasibilities in the solution, the primal (and possibly dual) objective value and the solution status.

Following parameters can be used to configure the printed statistics:

- *iparam.ana_sol_basis* enables or disables printing of statistics specific to the basis solution (condition number, number of basic variables etc.). Default is on.
- *iparam.ana_sol_print_violated* enables or disables listing names of all constraints (both primal and dual) which are violated by the solution. Default is off.
- *dparam.ana_sol_infeas_tol* is the tolerance defining when a constraint is considered violated. If a constraint is violated more than this, it will be listed in the summary.

Parameters

- *whichstream* (*mosek.streamtype*) – Index of the stream. (input)
- *whichsol* (*mosek.soltype*) – Selects a solution. (input)

Groups *Task diagnostics***Task.appendbarvars**

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

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

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

Groups *Symmetric matrix variable data***Task.appendcone**

```
void appendcone
(mosek.conetype ct,
 double conepr,
 int[] 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

$$\hat{x} = x_{\text{submem}[0]}, \dots, x_{\text{submem}[\text{nummem}-1]}.$$

Depending on the value of `ct` this function appends one of the constraints:

- 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 5.3.

Parameters

- `ct` (`mosek.conetype`) – Specifies the type of the cone. (input)
- `conepar` (`double`) – This argument is currently not used. It can be set to 0 (input)
- `submem` (`int[]`) – Variable subscripts of the members in the cone. (input)

Groups *Conic constraint data*

`Task.appendconeseq`

```
void appendconeseq
(mosek.conetype ct,
 double conepar,
 int nummem,
 int j)
```

Appends a new conic constraint to the problem, as in `Task.appendcone`. The function assumes the members of cone are sequential where the first member has index `j` and the last `j+nummem-1`.

Parameters

- `ct` (`mosek.conetype`) – Specifies the type of the cone. (input)
- `conepar` (`double`) – This argument is currently not used. It can be set to 0 (input)
- `nummem` (`int`) – Number of member variables in the cone. (input)
- `j` (`int`) – Index of the first variable in the conic constraint. (input)

Groups *Conic constraint data*

`Task.appendconesseq`


```
void appendconesseq
(mosek.conetype[] ct,
 double[] coneapar,
 int[] nummem,
 int j)
```

Appends a number of conic constraints to the problem, as in *Task.appendcone*. The k th cone is assumed to be of dimension `nummem[k]`. Moreover, it is assumed that the first variable of the first cone has index j and starting from there the sequentially following variables belong to the first cone, then to the second cone and so on.

Parameters

- `ct` (*mosek.conetype* []) – Specifies the type of the cone. (input)
- `coneapar` (*double* []) – This argument is currently not used. It can be set to 0 (input)
- `nummem` (*int* []) – Numbers of member variables in the cones. (input)
- `j` (*int*) – Index of the first variable in the first cone to be appended. (input)

Groups *Conic constraint data*

`Task.appendcons`

```
void appendcons (int num)
```

Appends a number of constraints to the model. Appended constraints will be declared free. Please note that **MOSEK** will automatically expand the problem dimension to accommodate the additional constraints.

Parameters `num` (*int*) – Number of constraints which should be appended. (input)

Groups *Linear constraint data*

`Task.appendsparsesymmat`

```
void appendsparsesymmat
(int dim,
 int[] subi,
 int[] subj,
 double[] valij,
 long[] idx)
```

```
long appendsparsesymmat
(int dim,
 int[] subi,
 int[] subj,
 double[] valij)
```

MOSEK maintains a storage of symmetric data matrices that is used to build \bar{C} and \bar{A} . The storage can be thought of as a vector of symmetric matrices denoted E . Hence, E_i is a symmetric matrix of certain dimension.

This function appends a general sparse symmetric matrix on triplet form to the vector E of symmetric matrices. The vectors `subi`, `subj`, and `valij` contains the row subscripts, column subscripts and values of each element in the symmetric matrix to be appended. Since the matrix that is appended is symmetric, only the lower triangular part should be specified. Moreover, duplicates are not allowed.

Observe the function reports the index (position) of the appended matrix in E . This index should be used for later references to the appended matrix.

Parameters

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

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

Groups *Symmetric matrix variable data*

Task.appendvars

```
void appendvars (int num)
```

Appends a number of variables to the model. Appended variables will be fixed at zero. Please note that **MOSEK** will automatically expand the problem dimension to accommodate the additional variables.

Parameters `num` (`int`) – Number of variables which should be appended. (input)

Groups *Scalar variable data*

Task.asyncgetresult

```
void asyncgetresult  
(String server,  
String port,  
String token,  
boolean[] respavailable,  
mosek.rescode[] resp,  
mosek.rescode[] trm)
```

```
boolean asyncgetresult  
(String server,  
String port,  
String token,  
mosek.rescode[] resp,  
mosek.rescode[] trm)
```

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

Parameters

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

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

Task.asyncoptimize

```
void asyncoptimize
(String server,
String port,
StringBuffer token)
```

```
String asyncoptimize
(String server,
String 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

- **server** (String) – Name or IP address of the solver server (input)
- **port** (String) – Network port of the solver service (input)
- **token** (StringBuffer) – Returns the task token (output)

Return (String) – Returns the task token

Task.asyncpoll

```
void asyncpoll
(String server,
String port,
String token,
boolean[] respavailable,
mosek.rescode[] resp,
mosek.rescode[] trm)
```

```
boolean asyncpoll
(String server,
String port,
String token,
mosek.rescode[] resp,
mosek.rescode[] trm)
```

Requests information about the status of the remote job.

Parameters

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

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

Task.asyncstop

```
void asyncstop
(String server,
 String port,
 String token)
```

Request that the job identified by the **token** is terminated.

Parameters

- **server** (String) – Name or IP address of the solver server (input)
- **port** (String) – Network port of the solver service (input)
- **token** (String) – The task token (input)

Task.basiscond

```
void basiscond
(double[] nrmbasis,
 double[] nrminvbasis)
```

If a basic solution is available and it defines a nonsingular basis, then this function computes the 1-norm estimate of the basis matrix and a 1-norm estimate for the inverse of the basis matrix. The 1-norm estimates are computed using the method outlined in [Ste98], pp. 388-391.

By definition the 1-norm condition number of a matrix B is defined as

$$\kappa_1(B) := \|B\|_1 \|B^{-1}\|_1.$$

Moreover, the larger the condition number is the harder it is to solve linear equation systems involving B . Given estimates for $\|B\|_1$ and $\|B^{-1}\|_1$ it is also possible to estimate $\kappa_1(B)$.

Parameters

- **nrmbasis** (double *by reference*) – An estimate for the 1-norm of the basis. (output)
- **nrminvbasis** (double *by reference*) – An estimate for the 1-norm of the inverse of the basis. (output)

Groups *Basis matrix*

Task.checkconvexity

```
void 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.

Groups *Task diagnostics*

Task.checkmem

```
void checkmem
(String file,
 int line)
```

Checks the memory allocated by the task.

Parameters

- **file** (**String**) – File from which the function is called. (input)
- **line** (**int**) – Line in the file from which the function is called. (input)

Groups *Memory*

~~Task.chgbound~~ *Deprecated*

```
void chgbound
(mosek.accmode accmode,
 int i,
 int lower,
 int finite,
 double 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:

$$\text{new lower bound} = \begin{cases} -\infty, & \text{finite} = 0, \\ \text{value} & \text{otherwise.} \end{cases}$$

Otherwise if **lower** is zero, then

$$\text{new upper bound} = \begin{cases} \infty, & \text{finite} = 0, \\ \text{value} & \text{otherwise.} \end{cases}$$

Please note that this function automatically updates the bound key for bound, in particular, if the lower and upper bounds are identical, the bound key is changed to **fixed**.

Parameters

- **accmode** (*mosek.accmode*) – Defines if operations are performed row-wise (constraint-oriented) or column-wise (variable-oriented). (input)
- **i** (**int**) – Index of the constraint or variable for which the bounds should be changed. (input)
- **lower** (**int**) – If non-zero, then the lower bound is changed, otherwise the upper bound is changed. (input)
- **finite** (**int**) – If non-zero, then **value** is assumed to be finite. (input)
- **value** (**double**) – New value for the bound. (input)

Groups *Bound data*

~~Task.chgconbound~~

```
void chgconbound
(int i,
 int lower,
 int finite,
 double value)
```

Changes a bound for one constraint.

If **lower** is non-zero, then the lower bound is changed as follows:

$$\text{new lower bound} = \begin{cases} -\infty, & \text{finite} = 0, \\ \text{value} & \text{otherwise.} \end{cases}$$

Otherwise if `lower` is zero, then

$$\text{new upper bound} = \begin{cases} \infty, & \text{finite} = 0, \\ \text{value} & \text{otherwise.} \end{cases}$$

Please note that this function automatically updates the bound key for the bound, in particular, if the lower and upper bounds are identical, the bound key is changed to **fixed**.

Parameters

- `i` (`int`) – Index of the constraint for which the bounds should be changed. (input)
- `lower` (`int`) – If non-zero, then the lower bound is changed, otherwise the upper bound is changed. (input)
- `finite` (`int`) – If non-zero, then `value` is assumed to be finite. (input)
- `value` (`double`) – New value for the bound. (input)

Groups *Bound data*

Task.chgvarbound

```
void chgvarbound
(int j,
 int lower,
 int finite,
 double value)
```

Changes a bound for one variable.

If `lower` is non-zero, then the lower bound is changed as follows:

$$\text{new lower bound} = \begin{cases} -\infty, & \text{finite} = 0, \\ \text{value} & \text{otherwise.} \end{cases}$$

Otherwise if `lower` is zero, then

$$\text{new upper bound} = \begin{cases} \infty, & \text{finite} = 0, \\ \text{value} & \text{otherwise.} \end{cases}$$

Please note that this function automatically updates the bound key for the bound, in particular, if the lower and upper bounds are identical, the bound key is changed to **fixed**.

Parameters

- `j` (`int`) – Index of the variable for which the bounds should be changed. (input)
- `lower` (`int`) – If non-zero, then the lower bound is changed, otherwise the upper bound is changed. (input)
- `finite` (`int`) – If non-zero, then `value` is assumed to be finite. (input)
- `value` (`double`) – New value for the bound. (input)

Groups *Bound data*

Task.commitchanges

```
void commitchanges ()
```

Commits all cached problem changes to the task. It is usually not necessary to call this function explicitly since changes will be committed automatically when required.

Groups *Scalar variable data*

Task.deletesolution

```
void deletesolution (mosek.soltype whichsol)
```

Undefine a solution and free the memory it uses.

Parameters *whichsol* (*mosek.soltype*) – Selects a solution. (input)

Groups *Task management*

Task.dualsensitivity

```
void dualsensitivity
(int[] subj,
 double[] leftpricej,
 double[] rightpricej,
 double[] leftrangej,
 double[] rightrangej)
```

Calculates sensitivity information for objective coefficients. The indexes of the coefficients to analyze are

$$\{\text{subj}[i] \mid i = 0, \dots, \text{numj} - 1\}$$

The type of sensitivity analysis to perform (basis or optimal partition) is controlled by the parameter *iparam.sensitivity_type*.

For an example, please see Section 17.1.4.

Parameters

- *subj* (*int[]*) – Indexes of objective coefficients to analyze. (input)
- *leftpricej* (*double[]*) – *leftpricej[j]* is the left shadow price for the coefficient with index *subj[j]*. (output)
- *rightpricej* (*double[]*) – *rightpricej[j]* is the right shadow price for the coefficient with index *subj[j]*. (output)
- *leftrangej* (*double[]*) – *leftrangej[j]* is the left range β_1 for the coefficient with index *subj[j]*. (output)
- *rightrangej* (*double[]*) – *rightrangej[j]* is the right range β_2 for the coefficient with index *subj[j]*. (output)

Groups *Sensitivity analysis*

Task.getacol

```
void getacol
(int j,
 int[] nzj,
 int[] subj,
 double[] valj)
```

Obtains one column of *A* in a sparse format.

Parameters

- *j* (*int*) – Index of the column. (input)
- *nzj* (*int by reference*) – Number of non-zeros in the column obtained. (output)
- *subj* (*int[]*) – Row indices of the non-zeros in the column obtained. (output)
- *valj* (*double[]*) – Numerical values in the column obtained. (output)

Groups *Scalar variable data*

Task.getacolnumnz

```
void getacolnumnz
(int i,
 int[] nzj)
```

```
int getacolnumnz (int i)
```

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

Parameters

- **i** (int) – Index of the column. (input)
- **nzj** (int *by reference*) – Number of non-zeros in the j -th column of A . (output)

Return (int) – Number of non-zeros in the j -th column of A .

Groups *Scalar variable data*

Task.getacolslicetrip

```
void getacolslicetrip
(int first,
 int last,
 int[] subi,
 int[] subj,
 double[] val)
```

Obtains a sequence of columns from A in sparse triplet format. The function returns the content of all columns whose index j satisfies $\text{first} \leq j < \text{last}$. The triplets corresponding to nonzero entries are stored in the arrays **subi**, **subj** and **val**.

Parameters

- **first** (int) – Index of the first column in the sequence. (input)
- **last** (int) – Index of the last column in the sequence **plus one**. (input)
- **subi** (int[]) – Constraint subscripts. (output)
- **subj** (int[]) – Column subscripts. (output)
- **val** (double[]) – Values. (output)

Groups *Scalar variable data*

Task.getaij

```
void getaij
(int i,
 int j,
 double[] aij)
```

```
double getaij
(int i,
 int j)
```

Obtains a single coefficient in A .

Parameters

- **i** (int) – Row index of the coefficient to be returned. (input)

- `j` (int) – Column index of the coefficient to be returned. (input)
- `aij` (double *by reference*) – The required coefficient $a_{i,j}$. (output)

Return (double) – The required coefficient $a_{i,j}$.

Groups *Scalar variable data*

`Task.getapiecenumnz`

```
void getapiecenumnz
(int firsti,
 int lasti,
 int firstj,
 int lastj,
 int[] numnz)
```

```
int getapiecenumnz
(int firsti,
 int lasti,
 int firstj,
 int lastj)
```

Obtains the number non-zeros in a rectangular piece of A , i.e. the number of elements in the set

$$\{(i, j) : a_{i,j} \neq 0, \text{firsti} \leq i \leq \text{lasti} - 1, \text{firstj} \leq j \leq \text{lastj} - 1\}$$

This function is not an efficient way to obtain the number of non-zeros in one row or column. In that case use the function `Task.getarownumnz` or `Task.getacolnumnz`.

Parameters

- `firsti` (int) – Index of the first row in the rectangular piece. (input)
- `lasti` (int) – Index of the last row plus one in the rectangular piece. (input)
- `firstj` (int) – Index of the first column in the rectangular piece. (input)
- `lastj` (int) – Index of the last column plus one in the rectangular piece. (input)
- `numnz` (int *by reference*) – Number of non-zero A elements in the rectangular piece. (output)

Return (int) – Number of non-zero A elements in the rectangular piece.

`Task.getarow`

```
void getarow
(int i,
 int[] nzi,
 int[] subi,
 double[] vali)
```

Obtains one row of A in a sparse format.

Parameters

- `i` (int) – Index of the row. (input)
- `nzi` (int *by reference*) – Number of non-zeros in the row obtained. (output)
- `subi` (int[]) – Column indices of the non-zeros in the row obtained. (output)
- `vali` (double[]) – Numerical values of the row obtained. (output)

Groups *Scalar variable data*

Task.getarownumnz

```
void getarownumnz
(int i,
 int[] nzi)
```

```
int getarownumnz (int i)
```

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

Parameters

- **i** (int) – Index of the row. (input)
- **nzi** (int *by reference*) – Number of non-zeros in the i -th row of A . (output)

Return (int) – Number of non-zeros in the i -th row of A .

Groups *Scalar variable data*

Task.getarowslicetrip

```
void getarowslicetrip
(int first,
 int last,
 int[] subi,
 int[] subj,
 double[] val)
```

Obtains a sequence of rows from A in sparse triplet format. The function returns the content of all rows whose index i satisfies `first <= i < last`. The triplets corresponding to nonzero entries are stored in the arrays `subi`, `subj` and `val`.

Parameters

- **first** (int) – Index of the first row in the sequence. (input)
- **last** (int) – Index of the last row in the sequence **plus one**. (input)
- **subi** (int[]) – Constraint subscripts. (output)
- **subj** (int[]) – Column subscripts. (output)
- **val** (double[]) – Values. (output)

Groups *Scalar variable data*

Task.getaslice

```
void getaslice
(mosek.accmode accmode,
 int first,
 int last,
 int[] ptrb,
 int[] ptre,
 int[] sub,
 double[] val)
```

```
void getaslice
(mosek.accmode accmode,
 int first,
 int last,
 long[] ptrb,
 long[] ptre,
```

```
int[] sub,
double[] val)
```

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

Parameters

- `accmode` (*mosek.accmode*) – Defines whether a column slice or a row slice is requested. (input)
- `first` (`int`) – Index of the first row or column in the sequence. (input)
- `last` (`int`) – Index of the last row or column in the sequence **plus one**. (input)
- `ptrb` (`int[]`) – `ptrb[t]` is an index pointing to the first element in the t -th row or column obtained. (output)
- `ptrb` (`long[]`) – `ptrb[t]` is an index pointing to the first element in the t -th row or column obtained. (output)
- `ptre` (`int[]`) – `ptre[t]` is an index pointing to the last element plus one in the t -th row or column obtained. (output)
- `ptre` (`long[]`) – `ptre[t]` is an index pointing to the last element plus one in the t -th row or column obtained. (output)
- `sub` (`int[]`) – Contains the row or column subscripts. (output)
- `val` (`double[]`) – Contains the coefficient values. (output)

Groups *Scalar variable data*

Task.getaslicenumnz

```
void getaslicenumnz
(mosek.accmode accmode,
 int first,
 int last,
 long[] numnz)
```

```
long getaslicenumnz
(mosek.accmode accmode,
 int first,
 int last)
```

Obtains the number of non-zeros in a slice of rows or columns of A .

Parameters

- `accmode` (*mosek.accmode*) – Defines whether non-zeros are counted in a column slice or a row slice. (input)
- `first` (`int`) – Index of the first row or column in the sequence. (input)
- `last` (`int`) – Index of the last row or column **plus one** in the sequence. (input)
- `numnz` (`long` *by reference*) – Number of non-zeros in the slice. (output)

Return (`long`) – Number of non-zeros in the slice.

Task.getbarablocktriplet

```
void getbarablocktriplet
(long[] num,
 int[] subi,
 int[] subj,
```

```
int[] subk,  
int[] subl,  
double[] valijkl)
```

```
long getbarablocktriplet  
(int[] subi,  
int[] subj,  
int[] subk,  
int[] subl,  
double[] valijkl)
```

Obtains \bar{A} in block triplet form.

Parameters

- num (long *by reference*) – Number of elements in the block triplet form. (output)
- subi (int[]) – Constraint index. (output)
- subj (int[]) – Symmetric matrix variable index. (output)
- subk (int[]) – Block row index. (output)
- subl (int[]) – Block column index. (output)
- valijkl (double[]) – The numerical value associated with each block triplet. (output)

Return (long) – Number of elements in the block triplet form.

Groups *Symmetric matrix variable data*

Task.getbaraidx

```
void getbaraidx  
(long idx,  
int[] i,  
int[] j,  
long[] num,  
long[] sub,  
double[] weights)
```

```
long getbaraidx  
(long idx,  
int[] i,  
int[] j,  
long[] sub,  
double[] weights)
```

Obtains information about an element in \bar{A} . Since \bar{A} is a sparse matrix of symmetric matrices, only the nonzero elements in \bar{A} are stored in order to save space. Now \bar{A} is stored vectorized i.e. as one long vector. This function makes it possible to obtain information such as the row index and the column index of a particular element of the vectorized form of \bar{A} .

Please observe if one element of \bar{A} is inputted multiple times then it may be stored several times in vectorized form. In that case the element with the highest index is the one that is used.

Parameters

- idx (long) – Position of the element in the vectorized form. (input)
- i (int *by reference*) – Row index of the element at position idx. (output)
- j (int *by reference*) – Column index of the element at position idx. (output)

- **num** (long *by reference*) – Number of terms in weighted sum that forms the element. (output)
- **sub** (long[]) – A list indexes of the elements from symmetric matrix storage that appear in the weighted sum. (output)
- **weights** (double[]) – The weights associated with each term in the weighted sum. (output)

Return (long) – Number of terms in weighted sum that forms the element.

Groups *Symmetric matrix variable data*

Task.getbaraidxij

```
void getbaraidxij
(long idx,
 int[] i,
 int[] j)
```

Obtains information about an element in \bar{A} . Since \bar{A} is a sparse matrix of symmetric matrices, only the nonzero elements in \bar{A} are stored in order to save space. Now \bar{A} is stored vectorized i.e. as one long vector. This function makes it possible to obtain information such as the row index and the column index of a particular element of the vectorized form of \bar{A} .

Please note that if one element of \bar{A} is inputted multiple times then it may be stored several times in vectorized form. In that case the element with the highest index is the one that is used.

Parameters

- **idx** (long) – Position of the element in the vectorized form. (input)
- **i** (int *by reference*) – Row index of the element at position **idx**. (output)
- **j** (int *by reference*) – Column index of the element at position **idx**. (output)

Groups *Symmetric matrix variable data*

Task.getbaraidxinfo

```
void getbaraidxinfo
(long idx,
 long[] num)
```

```
long getbaraidxinfo (long idx)
```

Each nonzero element in \bar{A}_{ij} is formed as a weighted sum of symmetric matrices. Using this function the number of terms in the weighted sum can be obtained. See description of *Task.appendsparsesymmat* for details about the weighted sum.

Parameters

- **idx** (long) – The internal position of the element for which information should be obtained. (input)
- **num** (long *by reference*) – Number of terms in the weighted sum that form the specified element in \bar{A} . (output)

Return (long) – Number of terms in the weighted sum that form the specified element in \bar{A} .

Groups *Symmetric matrix variable data*

Task.getbarasparsity

```
void getbarasparsity
(long[] numnz,
 long[] idxij)
```

The matrix \bar{A} is assumed to be a sparse matrix of symmetric matrices. This implies that many of the elements in \bar{A} are likely to be zero matrices. Therefore, in order to save space, only nonzero elements in \bar{A} are stored on vectorized form. This function is used to obtain the sparsity pattern of \bar{A} and the position of each nonzero element in the vectorized form of \bar{A} . From the index detailed information about each nonzero $\bar{A}_{i,j}$ can be obtained using *Task.getbaraidxinfo* and *Task.getbaraidx*.

Parameters

- numnz (long *by reference*) – Number of nonzero elements in \bar{A} . (output)
- idxij (long[]) – Position of each nonzero element in the vectorized form of \bar{A} . (output)

Groups *Symmetric matrix variable data*

Task.getbarcblocktriplet

```
void getbarcblocktriplet
(long[] num,
 int[] subj,
 int[] subk,
 int[] subl,
 double[] valjkl)
```

```
long getbarcblocktriplet
(int[] subj,
 int[] subk,
 int[] subl,
 double[] valjkl)
```

Obtains \bar{C} in block triplet form.

Parameters

- num (long *by reference*) – Number of elements in the block triplet form. (output)
- subj (int[]) – Symmetric matrix variable index. (output)
- subk (int[]) – Block row index. (output)
- subl (int[]) – Block column index. (output)
- valjkl (double[]) – The numerical value associated with each block triplet. (output)

Return (long) – Number of elements in the block triplet form.

Groups *Symmetric matrix variable data*

Task.getbarcidx

```
void getbarcidx
(long idx,
 int[] j,
 long[] num,
 long[] sub,
 double[] weights)
```

Obtains information about an element in \bar{C} .

Parameters

- `idx` (`long`) – Index of the element for which information should be obtained. (input)
- `j` (`int` *by reference*) – Row index in \overline{C} . (output)
- `num` (`long` *by reference*) – Number of terms in the weighted sum. (output)
- `sub` (`long[]`) – Elements appearing the weighted sum. (output)
- `weights` (`double[]`) – Weights of terms in the weighted sum. (output)

Groups *Symmetric matrix variable data*`Task.getbarcidxinfo`

```
void getbarcidxinfo
(long idx,
 long[] num)
```

```
long getbarcidxinfo (long idx)
```

Obtains the number of terms in the weighted sum that forms a particular element in \overline{C} .**Parameters**

- `idx` (`long`) – Index of the element for which information should be obtained. The value is an index of a symmetric sparse variable. (input)
- `num` (`long` *by reference*) – Number of terms that appear in the weighted sum that forms the requested element. (output)

Return (`long`) – Number of terms that appear in the weighted sum that forms the requested element.**Groups** *Symmetric matrix variable data*`Task.getbarcidxj`

```
void getbarcidxj
(long idx,
 int[] j)
```

Obtains the row index of an element in \overline{C} .**Parameters**

- `idx` (`long`) – Index of the element for which information should be obtained. (input)
- `j` (`int` *by reference*) – Row index in \overline{C} . (output)

Groups *Symmetric matrix variable data*`Task.getbarcsparsity`

```
void getbarcsparsity
(long[] numnz,
 long[] idxj)
```

Internally only the nonzero elements of \overline{C} are stored in a vector. This function is used to obtain the nonzero elements of \overline{C} and their indexes in the internal vector representation (in `idx`). From the index detailed information about each nonzero \overline{C}_j can be obtained using `Task.getbarcidxinfo` and `Task.getbarcidx`.

Parameters

- numnz (long *by reference*) – Number of nonzero elements in \overline{C} . (output)
- idxj (long[]) – Internal positions of the nonzero elements in \overline{C} . (output)

Groups *Symmetric matrix variable data*

Task.getbarsj

```
void getbarsj
(mosek.soltype whichsol,
 int j,
 double[] barsj)
```

Obtains the dual solution for a semidefinite variable. Only the lower triangular part of \overline{S}_j is returned because the matrix by construction is symmetric. The format is that the columns are stored sequentially in the natural order.

Parameters

- whichsol (*mosek.soltype*) – Selects a solution. (input)
- j (int) – Index of the semidefinite variable. (input)
- barsj (double[]) – Value of \overline{S}_j . (output)

Groups *Solution (get)*

Task.getbarvarname

```
void getbarvarname
(int i,
 StringBuffer name)
```

```
String getbarvarname (int i)
```

Obtains the name of a semidefinite variable.

Parameters

- i (int) – Index of the variable. (input)
- name (StringBuffer) – The requested name is copied to this buffer. (output)

Return (String) – The requested name is copied to this buffer.**Groups** *Naming*

Task.getbarvarnameindex

```
void getbarvarnameindex
(String somename,
 int[] asgn,
 int[] index)
```

```
int getbarvarnameindex
(String somename,
 int[] asgn)
```

Obtains the index of semidefinite variable from its name.

Parameters

- somename (String) – The name of the variable. (input)

- `asgn (int by reference)` – Non-zero if the name `somename` is assigned to some semidefinite variable. (output)
- `index (int by reference)` – The index of a semidefinite variable with the name `somename` (if one exists). (output)

Return (int) – The index of a semidefinite variable with the name `somename` (if one exists).

Groups *Naming*

`Task.getbarvarnamelen`

```
void getbarvarnamelen
(int i,
 int[] len)
```

```
int getbarvarnamelen (int i)
```

Obtains the length of the name of a semidefinite variable.

Parameters

- `i (int)` – Index of the variable. (input)
- `len (int by reference)` – Returns the length of the indicated name. (output)

Return (int) – Returns the length of the indicated name.

Groups *Naming*

`Task.getbarxj`

```
void getbarxj
(mosek.soltype whichsol,
 int j,
 double[] barxj)
```

Obtains the primal solution for a semidefinite variable. Only the lower triangular 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

- `whichsol (mosek.soltype)` – Selects a solution. (input)
- `j (int)` – Index of the semidefinite variable. (input)
- `barxj (double[])` – Value of \bar{X}_j . (output)

Groups *Solution (get)*

`Task.getbound`

```
void getbound
(mosek.acemode accmode,
 int i,
 mosek.boundkey[] bk,
 double[] bl,
 double[] bu)
```

Obtains bound information for one constraint or variable.

Parameters

- `accmode` (*mosek.accmode*) – Defines if operations are performed row-wise (constraint-oriented) or column-wise (variable-oriented). (input)
- `i` (`int`) – Index of the constraint or variable for which the bound information should be obtained. (input)
- `bk` (*mosek.boundkey by reference*) – Bound keys. (output)
- `bl` (`double by reference`) – Values for lower bounds. (output)
- `bu` (`double by reference`) – Values for upper bounds. (output)

Groups *Bound data*

`Task.getboundslice`

```
void getboundslice
(mosek.accmode accmode,
 int first,
 int last,
 mosek.boundkey[] bk,
 double[] bl,
 double[] bu)
```

Obtains bounds information for a slice of variables or constraints.

Parameters

- `accmode` (*mosek.accmode*) – Defines if operations are performed row-wise (constraint-oriented) or column-wise (variable-oriented). (input)
- `first` (`int`) – First index in the sequence. (input)
- `last` (`int`) – Last index plus 1 in the sequence. (input)
- `bk` (*mosek.boundkey []*) – Bound keys. (output)
- `bl` (`double[]`) – Values for lower bounds. (output)
- `bu` (`double[]`) – Values for upper bounds. (output)

Groups *Bound data*

`Task.getc`

```
void getc (double[] c)
```

Obtains all objective coefficients *c*.

Parameters `c` (`double[]`) – Linear terms of the objective as a dense vector. The length is the number of variables. (output)

Groups *Scalar variable data*

`Task.getcfix`

```
void getcfix (double[] cfix)
```

```
double getcfix ()
```

Obtains the fixed term in the objective.

Parameters `cfix` (`double by reference`) – Fixed term in the objective. (output)

Return (`double`) – Fixed term in the objective.

Groups *Scalar variable data*

Task.getcj

```
void getcj
(int j,
 double[] cj)
```

Obtains one coefficient of c .

Parameters

- j (int) – Index of the variable for which the c coefficient should be obtained. (input)
- cj (double *by reference*) – The value of c_j . (output)

Groups *Scalar variable data*

Task.getconbound

```
void getconbound
(int i,
 mosek.boundkey[] bk,
 double[] bl,
 double[] bu)
```

Obtains bound information for one constraint.

Parameters

- i (int) – Index of the constraint for which the bound information should be obtained. (input)
- bk (*mosek.boundkey by reference*) – Bound keys. (output)
- bl (double *by reference*) – Values for lower bounds. (output)
- bu (double *by reference*) – Values for upper bounds. (output)

Groups *Bound data*

Task.getconboundslice

```
void getconboundslice
(int first,
 int last,
 mosek.boundkey[] bk,
 double[] bl,
 double[] bu)
```

Obtains bounds information for a slice of the constraints.

Parameters

- $first$ (int) – First index in the sequence. (input)
- $last$ (int) – Last index plus 1 in the sequence. (input)
- bk (*mosek.boundkey []*) – Bound keys. (output)
- bl (double []) – Values for lower bounds. (output)
- bu (double []) – Values for upper bounds. (output)

Groups *Bound data*

Task.getcone

```
void getcone
(int k,
 mosek.conetype[] ct,
 double[] coneapar,
 int[] nummem,
 int[] submem)
```

Obtains a cone.

Parameters

- **k** (**int**) – Index of the cone. (input)
- **ct** (*mosek.conetype by reference*) – Specifies the type of the cone. (output)
- **coneapar** (**double by reference**) – This argument is currently not used. It can be set to 0 (output)
- **nummem** (**int by reference**) – Number of member variables in the cone. (output)
- **submem** (**int []**) – Variable subscripts of the members in the cone. (output)

Groups *Conic constraint data*

Task.getconeinfo

```
void getconeinfo
(int k,
 mosek.conetype[] ct,
 double[] coneapar,
 int[] nummem)
```

Obtains information about a cone.

Parameters

- **k** (**int**) – Index of the cone. (input)
- **ct** (*mosek.conetype by reference*) – Specifies the type of the cone. (output)
- **coneapar** (**double by reference**) – This argument is currently not used. It can be set to 0 (output)
- **nummem** (**int by reference**) – Number of member variables in the cone. (output)

Groups *Conic constraint data*

Task.getconename

```
void getconename
(int i,
 StringBuffer name)
```

```
String getconename (int i)
```

Obtains the name of a cone.

Parameters

- **i** (**int**) – Index of the cone. (input)
- **name** (**StringBuffer**) – The required name. (output)

Return (**String**) – The required name.

Groups *Naming*

Task.getconenameindex

```
void getconenameindex
(String somename,
 int[] asgn,
 int[] index)
```

```
int getconenameindex
(String somename,
 int[] asgn)
```

Checks whether the name **somename** has been assigned to any cone. If it has been assigned to a cone, then the index of the cone is reported.

Parameters

- **somename** (**String**) – The name which should be checked. (input)
- **asgn** (**int** *by reference*) – Is non-zero if the name **somename** is assigned to some cone. (output)
- **index** (**int** *by reference*) – If the name **somename** is assigned to some cone, then **index** is the index of the cone. (output)

Return (**int**) – If the name **somename** is assigned to some cone, then **index** is the index of the cone.

Groups *Naming*

Task.getconenamelen

```
void getconenamelen
(int i,
 int[] len)
```

```
int getconenamelen (int i)
```

Obtains the length of the name of a cone.

Parameters

- **i** (**int**) – Index of the cone. (input)
- **len** (**int** *by reference*) – Returns the length of the indicated name. (output)

Return (**int**) – Returns the length of the indicated name.

Groups *Naming*

Task.getconname

```
void getconname
(int i,
 StringBuffer name)
```

```
String getconname (int i)
```

Obtains the name of a constraint.

Parameters

- **i** (**int**) – Index of the constraint. (input)
- **name** (**StringBuffer**) – The required name. (output)

Return (String) – The required name.

Groups *Naming*

Task.getconnameindex

```
void getconnameindex
(String somename,
 int[] asgn,
 int[] index)
```

```
int getconnameindex
(String somename,
 int[] asgn)
```

Checks whether the name **somename** has been assigned to any constraint. If so, the index of the constraint is reported.

Parameters

- **somename** (String) – The name which should be checked. (input)
- **asgn** (int *by reference*) – Is non-zero if the name **somename** is assigned to some constraint. (output)
- **index** (int *by reference*) – If the name **somename** is assigned to a constraint, then **index** is the index of the constraint. (output)

Return (int) – If the name **somename** is assigned to a constraint, then **index** is the index of the constraint.

Groups *Naming*

Task.getconnamelen

```
void getconnamelen
(int i,
 int[] len)
```

```
int getconnamelen (int i)
```

Obtains the length of the name of a constraint.

Parameters

- **i** (int) – Index of the constraint. (input)
- **len** (int *by reference*) – Returns the length of the indicated name. (output)

Return (int) – Returns the length of the indicated name.

Groups *Naming*

Task.getcslice

```
void getcslice
(int first,
 int last,
 double[] c)
```

Obtains a sequence of elements in *c*.

Parameters

- **first** (int) – First index in the sequence. (input)

- `last (int)` – Last index plus 1 in the sequence. (input)
- `c (double[])` – Linear terms of the requested slice of the objective as a dense vector. The length is `last-first`. (output)

Groups *Scalar variable data*

`Task.getdimbarvarj`

```
void getdimbarvarj
(int j,
 int[] dimbarvarj)
```

```
int getdimbarvarj (int j)
```

Obtains the dimension of a symmetric matrix variable.

Parameters

- `j (int)` – Index of the semidefinite variable whose dimension is requested. (input)
- `dimbarvarj (int by reference)` – The dimension of the j -th semidefinite variable. (output)

Return `(int)` – The dimension of the j -th semidefinite variable.

Groups *Symmetric matrix variable data*

`Task.getdouinf`

```
void getdouinf
(mosek.dinfitem whichdinf,
 double[] dvalue)
```

```
double getdouinf (mosek.dinfitem whichdinf)
```

Obtains a double information item from the task information database.

Parameters

- `whichdinf (mosek.dinfitem)` – Specifies a double information item. (input)
- `dvalue (double by reference)` – The value of the required double information item. (output)

Return `(double)` – The value of the required double information item.

Groups *Optimizer statistics*

`Task.getdouparam`

```
void getdouparam
(mosek.dparam param,
 double[] parvalue)
```

```
double getdouparam (mosek.dparam param)
```

Obtains the value of a double parameter.

Parameters

- `param (mosek.dparam)` – Which parameter. (input)
- `parvalue (double by reference)` – Parameter value. (output)

Return (double) – Parameter value.

Groups *Parameters (get)*

Task.getdualobj

```
void getdualobj
(mosek.soltype whichsol,
 double[] dualobj)
```

Computes the dual objective value associated with the solution. Note that if the solution is a primal infeasibility certificate, then the fixed term in the objective value is not included.

Moreover, since there is no dual solution associated with an integer solution, an error will be reported if the dual objective value is requested for the integer solution.

Parameters

- whichsol (*mosek.soltype*) – Selects a solution. (input)
- dualobj (double *by reference*) – Objective value corresponding to the dual solution. (output)

Groups *Solution information*

Task.getdualsolutionnorms

```
void getdualsolutionnorms
(mosek.soltype whichsol,
 double[] nrmy,
 double[] nrmslc,
 double[] nrmsuc,
 double[] nrmslx,
 double[] nrmsux,
 double[] nrmsnx,
 double[] nrmbars)
```

Compute norms of the dual solution.

Parameters

- whichsol (*mosek.soltype*) – Selects a solution. (input)
- nrmy (double *by reference*) – The norm of the y vector. (output)
- nrmslc (double *by reference*) – The norm of the s_l^c vector. (output)
- nrmsuc (double *by reference*) – The norm of the s_u^c vector. (output)
- nrmslx (double *by reference*) – The norm of the s_l^x vector. (output)
- nrmsux (double *by reference*) – The norm of the s_u^x vector. (output)
- nrmsnx (double *by reference*) – The norm of the s_n^x vector. (output)
- nrmbars (double *by reference*) – The norm of the \bar{S} vector. (output)

Groups *Solution information*

Task.getdviolbarvar

```
void getdviolbarvar
(mosek.soltype whichsol,
 int[] sub,
 double[] 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 and when it is dual feasible solution the violation should be small.

Parameters

- **whichsol** (*mosek.soltype*) – Selects a solution. (input)
- **sub** (int[]) – An array of indexes of \bar{X} variables. (input)
- **viol** (double[]) – **viol[k]** is the violation of the solution for the constraint $\bar{S}_{\text{sub}[k]} \in \mathcal{S}_+$. (output)

Groups *Solution information*

Task.getdviolcon

```
void getdviolcon
(mosek.soltype whichsol,
 int[] sub,
 double[] viol)
```

The violation of the dual solution associated with the i -th constraint is computed as follows

$$\max(\rho((s_l^c)_i^*, (b_l^c)_i), \rho((s_u^c)_i^*, -(b_u^c)_i), | -y_i + (s_l^c)_i^* - (s_u^c)_i^* |)$$

where

$$\rho(x, l) = \begin{cases} -x, & l > -\infty, \\ |x|, & \text{otherwise.} \end{cases}$$

Both when the solution is a certificate of primal infeasibility or it is a dual feasible solution the violation should be small.

Parameters

- **whichsol** (*mosek.soltype*) – Selects a solution. (input)
- **sub** (int[]) – An array of indexes of constraints. (input)
- **viol** (double[]) – **viol[k]** is the violation of dual solution associated with the constraint **sub[k]**. (output)

Groups *Solution information*

Task.getdviolcones

```
void getdviolcones
(mosek.soltype whichsol,
 int[] sub,
 double[] viol)
```

Let $(s_n^x)^*$ be the value of variable (s_n^x) for the specified solution. For simplicity let us assume that s_n^x is a member of a quadratic cone, then the violation is computed as follows

$$\begin{cases} \max(0, (\|s_n^x\|_{2:n}^* - (s_n^x)_1^*)/\sqrt{2}), & (s_n^x)^* \geq -\|(s_n^x)_{2:n}^*\|, \\ \|(s_n^x)^*\|, & \text{otherwise.} \end{cases}$$

Both when the solution is a certificate of primal infeasibility or when it is a dual feasible solution the violation should be small.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `sub` (`int[]`) – An array of indexes of conic constraints. (input)
- `viol` (`double[]`) – `viol[k]` is the violation of the dual solution associated with the conic constraint `sub[k]`. (output)

Groups *Solution information*

`Task.getdviolvar`

```
void getdviolvar
(mosek.soltype whichsol,
 int[] sub,
 double[] viol)
```

The violation of the dual solution associated with the j -th variable is computed as follows

$$\max \left(\rho((s_l^x)_j^*, (b_l^x)_j), \rho((s_u^x)_j^*, -(b_u^x)_j), \left| \sum_{i=0}^{numcon-1} a_{ij}y_i + (s_l^x)_j^* - (s_u^x)_j^* - \tau c_j \right| \right)$$

where

$$\rho(x, l) = \begin{cases} -x, & l > -\infty, \\ |x|, & \text{otherwise} \end{cases}$$

and $\tau = 0$ if the solution is a certificate of primal infeasibility and $\tau = 1$ otherwise. The formula for computing the violation is only shown for the linear case but is generalized appropriately for the more general problems. Both when the solution is a certificate of primal infeasibility or when it is a dual feasible solution the violation should be small.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `sub` (`int[]`) – An array of indexes of x variables. (input)
- `viol` (`double[]`) – `viol[k]` is the violation of dual solution associated with the variable `sub[k]`. (output)

Groups *Solution information*

`Task.getinfeasiblesubproblem`

```
void getinfeasiblesubproblem
(mosek.soltype whichsol,
 Task[] inftask)
```

```
Task getinfeasiblesubproblem (mosek.soltype whichsol)
```

Given the solution is a certificate of primal or dual infeasibility then a primal or dual infeasible subproblem is obtained respectively. The subproblem tends to be much smaller than the original problem and hence it is easier to locate the infeasibility inspecting the subproblem than the original problem.

For the procedure to be useful it is important to assign meaningful names to constraints, variables etc. in the original task because those names will be duplicated in the subproblem.

The function is only applicable to linear and conic quadratic optimization problems.

For more information see Section 16.

Parameters

- `whichsol` (*mosek.soltype*) – Which solution to use when determining the infeasible subproblem. (input)

- `inftask` (*Task by reference*) – A new task containing the infeasible subproblem. (output)

Return (*Task*) – A new task containing the infeasible subproblem.

Groups *Infeasibility diagnostics*

`Task.getinfindex`

```
void getinfindex
(mosek.inftype inftype,
 String infname,
 int[] infindex)
```

Obtains the index of a named information item.

Parameters

- `inftype` (*mosek.inftype*) – Type of the information item. (input)
- `infname` (`String`) – Name of the information item. (input)
- `infindex` (`int` *by reference*) – The item index. (output)

Groups *Optimizer statistics*

`Task.getinfmax`

```
void getinfmax
(mosek.inftype inftype,
 int[] infmax)
```

Obtains the maximum index of an information item of a given type `inftype` plus 1.

Parameters

- `inftype` (*mosek.inftype*) – Type of the information item. (input)
- `infmax` (`int[]`) – The maximum index (plus 1) requested. (output)

Groups *Optimizer statistics*

`Task.getinfname`

```
void getinfname
(mosek.inftype inftype,
 int whichinf,
 StringBuffer infname)
```

Obtains the name of an information item.

Parameters

- `inftype` (*mosek.inftype*) – Type of the information item. (input)
- `whichinf` (`int`) – An information item. (input)
- `infname` (`StringBuffer`) – Name of the information item. (output)

Groups *Optimizer statistics*

`Task.getintinf`

```
void getintinf
(mosek.iinfitem whichiinf,
 int[] ivalue)
```

```
int getintinf (mosek.iinfitem whichiinf)
```

Obtains an integer information item from the task information database.

Parameters

- `whichiinf` (*mosek.iinfitem*) – Specifies an integer information item. (input)
- `ivalue` (*int by reference*) – The value of the required integer information item. (output)

Return (*int*) – The value of the required integer information item.

Groups *Optimizer statistics*

`Task.getintparam`

```
void getintparam  
(mosek.iparam param,  
 int[] parvalue)
```

```
int getintparam (mosek.iparam param)
```

Obtains the value of an integer parameter.

Parameters

- `param` (*mosek.iparam*) – Which parameter. (input)
- `parvalue` (*int by reference*) – Parameter value. (output)

Return (*int*) – Parameter value.

Groups *Parameters (get)*

`Task.getlenbarvarj`

```
void getlenbarvarj  
(int j,  
 long[] lenbarvarj)
```

```
long getlenbarvarj (int j)
```

Obtains the length of the j -th semidefinite variable i.e. the number of elements in the lower triangular part.

Parameters

- `j` (*int*) – Index of the semidefinite variable whose length if requested. (input)
- `lenbarvarj` (*long by reference*) – Number of scalar elements in the lower triangular part of the semidefinite variable. (output)

Return (*long*) – Number of scalar elements in the lower triangular part of the semidefinite variable.

Groups *Scalar variable data*

`Task.getlintinf`

```
void getlintinf  
(mosek.liinfitem whichliinf,  
 long[] ivalue)
```

```
long getlintinf (mosek.liinfitem whichliinf)
```

Obtains a long integer information item from the task information database.

Parameters

- **whichliinf** (*mosek.liinfitem*) – Specifies a long information item. (input)
- **ivalue** (long *by reference*) – The value of the required long integer information item. (output)

Return (long) – The value of the required long integer information item.

Groups *Optimizer statistics*

Task.getMaxnumanz

```
void getMaxnumanz (long[] maxnumanz)
```

```
long 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 .

Parameters **maxnumanz** (long *by reference*) – Number of preallocated non-zero linear matrix elements. (output)

Return (long) – Number of preallocated non-zero linear matrix elements.

Groups *Scalar variable data*

Task.getMaxnumbarvar

```
void getMaxnumbarvar (int[] maxnumbarvar)
```

```
int getMaxnumbarvar ()
```

Obtains maximum number of symmetric matrix variables for which space is currently preallocated.

Parameters **maxnumbarvar** (int *by reference*) – Maximum number of symmetric matrix variables for which space is currently preallocated. (output)

Return (int) – Maximum number of symmetric matrix variables for which space is currently preallocated.

Groups *Symmetric matrix variable data*

Task.getMaxnumcon

```
void getMaxnumcon (int[] maxnumcon)
```

Obtains the number of preallocated constraints in the optimization task. When this number of constraints is reached **MOSEK** will automatically allocate more space for constraints.

Parameters **maxnumcon** (int *by reference*) – Number of preallocated constraints in the optimization task. (output)

Groups *Linear constraint data*

Task.getMaxnumcone

```
void getMaxnumcone (int[] maxnumcone)
```

Obtains the number of preallocated cones in the optimization task. When this number of cones is reached **MOSEK** will automatically allocate space for more cones.

Parameters `maxnumcone` (*int by reference*) – Number of preallocated conic constraints in the optimization task. (output)

Groups *Task management*

`Task.getmaxnumqnz`

```
void getmaxnumqnz (long[] maxnumqnz)
```

Obtains the number of preallocated non-zeros for Q (both objective and constraints). When this number of non-zeros is reached **MOSEK** will automatically allocate more space for Q .

Parameters `maxnumqnz` (*long by reference*) – Number of non-zero elements preallocated in quadratic coefficient matrices. (output)

Groups *Scalar variable data*

`Task.getmaxnumvar`

```
void getmaxnumvar (int[] maxnumvar)
```

Obtains the number of preallocated variables in the optimization task. When this number of variables is reached **MOSEK** will automatically allocate more space for variables.

Parameters `maxnumvar` (*int by reference*) – Number of preallocated variables in the optimization task. (output)

Groups *Scalar variable data*

`Task.getmemusage`

```
void getmemusage  
(long[] meminuse,  
 long[] maxmemuse)
```

Obtains information about the amount of memory used by a task.

Parameters

- `meminuse` (*long by reference*) – Amount of memory currently used by the **task**. (output)
- `maxmemuse` (*long by reference*) – Maximum amount of memory used by the **task** until now. (output)

Groups *Memory*

`Task.getnumanz`

```
void getnumanz (int[] numanz)
```

```
int getnumanz ()
```

Obtains the number of non-zeros in A .

Parameters `numanz` (*int by reference*) – Number of non-zero elements in the linear constraint matrix. (output)

Return (*int*) – Number of non-zero elements in the linear constraint matrix.

Groups *Scalar variable data*

Task.getnumanz64

```
void getnumanz64 (long[] numanz)
```

```
long getnumanz64 ()
```

Obtains the number of non-zeros in A .

Parameters numanz (long *by reference*) – Number of non-zero elements in the linear constraint matrix. (output)

Return (long) – Number of non-zero elements in the linear constraint matrix.

Groups *Scalar variable data*

Task.getnumbarablocktriplets

```
void getnumbarablocktriplets (long[] num)
```

```
long getnumbarablocktriplets ()
```

Obtains an upper bound on the number of elements in the block triplet form of \bar{A} .

Parameters num (long *by reference*) – An upper bound on the number of elements in the block triplet form of \bar{A} . (output)

Return (long) – An upper bound on the number of elements in the block triplet form of \bar{A} .

Groups *Symmetric matrix variable data*

Task.getnumbaranz

```
void getnumbaranz (long[] nz)
```

```
long getnumbaranz ()
```

Get the number of nonzero elements in \bar{A} .

Parameters nz (long *by reference*) – The number of nonzero block elements in \bar{A} i.e. the number of \bar{A}_{ij} elements that are nonzero. (output)

Return (long) – The number of nonzero block elements in \bar{A} i.e. the number of \bar{A}_{ij} elements that are nonzero.

Groups *Symmetric matrix variable data*

Task.getnumbarcblocktriplets

```
void getnumbarcblocktriplets (long[] num)
```

```
long getnumbarcblocktriplets ()
```

Obtains an upper bound on the number of elements in the block triplet form of \bar{C} .

Parameters num (long *by reference*) – An upper bound on the number of elements in the block triplet form of \bar{C} . (output)

Return (long) – An upper bound on the number of elements in the block triplet form of \bar{C} .

Groups *Symmetric matrix variable data*

Task.getnumbarcnz

```
void getnumbarcnz (long[] nz)
```

```
long getnumbarcnz ()
```

Obtains the number of nonzero elements in \overline{C} .

Parameters `nz` (*long by reference*) – The number of nonzeros in \overline{C} i.e. the number of elements \overline{C}_j that are nonzero. (output)

Return (*long*) – The number of nonzeros in \overline{C} i.e. the number of elements \overline{C}_j that are nonzero.

Groups *Symmetric matrix variable data*

Task.getnumbarvar

```
void getnumbarvar (int[] numbarvar)
```

```
int getnumbarvar ()
```

Obtains the number of semidefinite variables.

Parameters `numbarvar` (*int by reference*) – Number of semidefinite variables in the problem. (output)

Return (*int*) – Number of semidefinite variables in the problem.

Groups *Symmetric matrix variable data*

Task.getnumcon

```
void getnumcon (int[] numcon)
```

```
int getnumcon ()
```

Obtains the number of constraints.

Parameters `numcon` (*int by reference*) – Number of constraints. (output)

Return (*int*) – Number of constraints.

Groups *Linear constraint data*

Task.getnumcone

```
void getnumcone (int[] numcone)
```

```
int getnumcone ()
```

Obtains the number of cones.

Parameters `numcone` (*int by reference*) – Number of conic constraints. (output)

Return (*int*) – Number of conic constraints.

Groups *Conic constraint data*

Task.getnumconemem


```
void getnumconmem
(int k,
 int[] nummem)
```

Obtains the number of members in a cone.

Parameters

- **k** (**int**) – Index of the cone. (input)
- **nummem** (**int** *by reference*) – Number of member variables in the cone. (output)

Groups *Conic constraint data*

Task.getnumintvar

```
void getnumintvar (int[] numintvar)
```

Obtains the number of integer-constrained variables.

Parameters **numintvar** (**int** *by reference*) – Number of integer variables. (output)

Groups *Scalar variable data*

Task.getnumparam

```
void getnumparam
(mosek.parametertype partype,
 int[] numparam)
```

Obtains the number of parameters of a given type.

Parameters

- **partype** (*mosek.parametertype*) – Parameter type. (input)
- **numparam** (**int** *by reference*) – The number of parameters of type **partype**. (output)

Groups *Parameter management*

Task.getnumqconknz

```
void getnumqconknz
(int k,
 long[] numqcnz)
```

```
long getnumqconknz (int k)
```

Obtains the number of non-zero quadratic terms in a constraint.

Parameters

- **k** (**int**) – Index of the constraint for which the number quadratic terms should be obtained. (input)
- **numqcnz** (**long** *by reference*) – Number of quadratic terms. (output)

Return (**long**) – Number of quadratic terms.

Groups *Scalar variable data*

Task.getnumqobjnz

```
void getnumqobjnz (long[] numqonz)
```

```
long getnumqobjnz ()
```

Obtains the number of non-zero quadratic terms in the objective.

Parameters numqonz (long *by reference*) – Number of non-zero elements in the quadratic objective terms. (output)

Return (long) – Number of non-zero elements in the quadratic objective terms.

Groups *Scalar variable data*

Task.getnumsymmat

```
void getnumsymmat (long[] num)
```

Obtains the number of symmetric matrices stored in the vector E .

Parameters num (long *by reference*) – The number of symmetric sparse matrices. (output)

Groups *Scalar variable data*

Task.getnumvar

```
void getnumvar (int[] numvar)
```

```
int getnumvar ()
```

Obtains the number of variables.

Parameters numvar (int *by reference*) – Number of variables. (output)

Return (int) – Number of variables.

Groups *Scalar variable data*

Task.getobjname

```
void getobjname (StringBuffer objname)
```

```
String getobjname ()
```

Obtains the name assigned to the objective function.

Parameters objname (StringBuffer) – Assigned the objective name. (output)

Return (String) – Assigned the objective name.

Groups *Naming*

Task.getobjnamelen

```
void getobjnamelen (int[] len)
```

```
int getobjnamelen ()
```

Obtains the length of the name assigned to the objective function.

Parameters len (int *by reference*) – Assigned the length of the objective name. (output)

Return (int) – Assigned the length of the objective name.

Groups *Naming*

Task.getobjsense

```
void getobjsense (mosek.objsense[] sense)
```

```
mosek.objsense getobjsense ()
```

Gets the objective sense of the task.

Parameters *sense* (*mosek.objsense by reference*) – The returned objective sense. (output)

Return (*mosek.objsense*) – The returned objective sense.

Groups *Objective data*

Task.getparammax

```
void getparammax  
(mosek.parametertype partype,  
 int[] parammax)
```

Obtains the maximum index of a parameter of type *partype* plus 1.

Parameters

- *partype* (*mosek.parametertype*) – Parameter type. (input)
- *parammax* (int *by reference*) – The maximum index (plus 1) of the given parameter type. (output)

Groups *Parameter management*

Task.getparamname

```
void getparamname  
(mosek.parametertype partype,  
 int param,  
 StringBuffer parname)
```

Obtains the name for a parameter *param* of type *partype*.

Parameters

- *partype* (*mosek.parametertype*) – Parameter type. (input)
- *param* (int) – Which parameter. (input)
- *parname* (StringBuffer) – Parameter name. (output)

Groups *Parameter management*

Task.getprimalobj

```
void getprimalobj  
(mosek.soltype whichsol,  
 double[] primalobj)
```

```
double getprimalobj (mosek.soltype whichsol)
```

Computes the primal objective value for the desired solution. Note that if the solution is an infeasibility certificate, then the fixed term in the objective is not included.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `primalobj` (double *by reference*) – Objective value corresponding to the primal solution. (output)

Return (double) – Objective value corresponding to the primal solution.

Groups *Solution information*

`Task.getprimalsolutionnorms`

```
void getprimalsolutionnorms
(mosek.soltype whichsol,
 double[] nrmxc,
 double[] nrmxx,
 double[] nrmbarx)
```

Compute norms of the primal solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `nrmxc` (double *by reference*) – The norm of the x^c vector. (output)
- `nrmxx` (double *by reference*) – The norm of the x vector. (output)
- `nrmbarx` (double *by reference*) – The norm of the \bar{X} vector. (output)

Groups *Solution information*

`Task.getprodtype`

```
void getprodtype (mosek.problemtypes[] probdtype)
```

```
mosek.problemtypes getprodtype ()
```

Obtains the problem type.

Parameters `prodtype` (*mosek.problemtypes by reference*) – The problem type. (output)

Return (*mosek.problemtypes*) – The problem type.

Groups *Task diagnostics*

`Task.getprosta`

```
void getprosta
(mosek.soltype whichsol,
 mosek.prosta[] prosta)
```

```
mosek.prosta getprosta (mosek.soltype whichsol)
```

Obtains the problem status.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `prosta` (*mosek.prosta by reference*) – Problem status. (output)

Return (*mosek.prosta*) – Problem status.

Groups *Solution information*

Task.getpviolbarvar

```
void getpviolbarvar
(mosek.soltype whichsol,
 int[] sub,
 double[] viol)
```

Computes the primal solution violation for a set of semidefinite variables. Let $(\bar{X}_j)^*$ be the value of the variable \bar{X}_j for the specified solution. Then the primal violation of the solution associated with variable \bar{X}_j is given by

$$\max(-\lambda_{\min}(\bar{X}_j), 0.0).$$

Both when the solution is a certificate of dual infeasibility or when it is primal feasible the violation should be small.

Parameters

- **whichsol** (*mosek.soltype*) – Selects a solution. (input)
- **sub** (int[]) – An array of indexes of \bar{X} variables. (input)
- **viol** (double[]) – **viol[k]** is how much the solution violates the constraint $\bar{X}_{\text{sub}[k]} \in \mathcal{S}_+$. (output)

Groups *Solution information*

Task.getpviolcon

```
void getpviolcon
(mosek.soltype whichsol,
 int[] sub,
 double[] viol)
```

Computes the primal solution violation for a set of constraints. The primal violation of the solution associated with the i -th constraint is given by

$$\max(\tau l_i^c - (x_i^c)^*, (x_i^c)^* - \tau u_i^c, \left| \sum_{j=0}^{\text{numvar}-1} a_{ij} x_j^* - x_i^c \right|)$$

where $\tau = 0$ if the solution is a certificate of dual infeasibility and $\tau = 1$ otherwise. Both when the solution is a certificate of dual infeasibility and when it is primal feasible the violation should be small. The above formula applies for the linear case but is appropriately generalized in other cases.

Parameters

- **whichsol** (*mosek.soltype*) – Selects a solution. (input)
- **sub** (int[]) – An array of indexes of constraints. (input)
- **viol** (double[]) – **viol[k]** is the violation associated with the solution for the constraint **sub[k]**. (output)

Groups *Solution information*

Task.getpviolcones

```
void getpviolcones
(mosek.soltype whichsol,
 int[] sub,
 double[] viol)
```

Computes the primal solution violation for a set of conic constraints. Let x^* be the value of the variable x for the specified solution. For simplicity let us assume that x is a member of a quadratic cone, then the violation is computed as follows

$$\begin{cases} \max(0, \|x_{2:n}\| - x_1)/\sqrt{2}, & x_1 \geq -\|x_{2:n}\|, \\ \|x\|, & \text{otherwise.} \end{cases}$$

Both when the solution is a certificate of dual infeasibility or when it is primal feasible the violation should be small.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `sub` (`int[]`) – An array of indexes of conic constraints. (input)
- `viol` (`double[]`) – `viol[k]` is the violation of the solution associated with the conic constraint number `sub[k]`. (output)

Groups *Solution information*

`Task.getpviolvar`

```
void getpviolvar
(mosek.soltype whichsol,
 int[] sub,
 double[] viol)
```

Computes the primal solution violation associated to a set of variables. Let x_j^* be the value of x_j for the specified solution. Then the primal violation of the solution associated with variable x_j is given by

$$\max(\tau l_j^x - x_j^*, x_j^* - \tau u_j^x, 0).$$

where $\tau = 0$ if the solution is a certificate of dual infeasibility and $\tau = 1$ otherwise. Both when the solution is a certificate of dual infeasibility and when it is primal feasible the violation should be small.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `sub` (`int[]`) – An array of indexes of x variables. (input)
- `viol` (`double[]`) – `viol[k]` is the violation associated with the solution for the variable $x_{\text{sub}[k]}$. (output)

Groups *Solution information*

`Task.getqconk`

```
void getqconk
(int k,
 long[] numqcnz,
 int[] qcsubi,
 int[] qcsubj,
 double[] qcval)
```

```
long getqconk
(int k,
 int[] qcsubi,
 int[] qcsubj,
 double[] qcval)
```

Obtains all the quadratic terms in a constraint. The quadratic terms are stored sequentially in qcsubi, qcsubj, and qcval.

Parameters

- k (int) – Which constraint. (input)
- numqcnz (long *by reference*) – Number of quadratic terms. (output)
- qcsubi (int[]) – Row subscripts for quadratic constraint matrix. (output)
- qcsubj (int[]) – Column subscripts for quadratic constraint matrix. (output)
- qcval (double[]) – Quadratic constraint coefficient values. (output)

Return (long) – Number of quadratic terms.

Groups *Scalar variable data*

Task.getqobj

```
void getqobj
(long[] numqonz,
 int[] qosubi,
 int[] qosubj,
 double[] qoval)
```

Obtains the quadratic terms in the objective. The required quadratic terms are stored sequentially in qosubi, qosubj, and qoval.

Parameters

- numqonz (long *by reference*) – Number of non-zero elements in the quadratic objective terms. (output)
- qosubi (int[]) – Row subscripts for quadratic objective coefficients. (output)
- qosubj (int[]) – Column subscripts for quadratic objective coefficients. (output)
- qoval (double[]) – Quadratic objective coefficient values. (output)

Groups *Scalar variable data*

Task.getqobjij

```
void getqobjij
(int i,
 int j,
 double[] qoij)
```

Obtains one coefficient q_{ij}^o in the quadratic term of the objective.

Parameters

- i (int) – Row index of the coefficient. (input)
- j (int) – Column index of coefficient. (input)
- qoij (double *by reference*) – The required coefficient. (output)

Groups *Scalar variable data*

Task.getreducedcosts

```
void getreducedcosts
(mosek.soltype whichsol,
 int first,
 int last,
 double[] redcosts)
```

Computes the reduced costs for a slice of variables and returns them in the array `redcosts` i.e.

$$\text{redcosts}[j - \text{first}] = (s_l^x)_j - (s_u^x)_j, \quad j = \text{first}, \dots, \text{last} - 1 \quad (18.2)$$

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (int) – The index of the first variable in the sequence. (input)
- `last` (int) – The index of the last variable in the sequence plus 1. (input)
- `redcosts` (double[]) – The reduced costs for the required slice of variables. (output)

Groups *Solution (get)*

Task.getskc

```
void getskc
(mosek.soltype whichsol,
 mosek.stakey[] skc)
```

Obtains the status keys for the constraints.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `skc` (*mosek.stakey*[]) – Status keys for the constraints. (output)

Groups *Solution (get)*

Task.getskcslice

```
void getskcslice
(mosek.soltype whichsol,
 int first,
 int last,
 mosek.stakey[] skc)
```

Obtains the status keys for a slice of the constraints.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (int) – First index in the sequence. (input)
- `last` (int) – Last index plus 1 in the sequence. (input)
- `skc` (*mosek.stakey*[]) – Status keys for the constraints. (output)

Groups *Solution (get)*

Task.getskx


```
void getskx
(mosek.soltype whichsol,
 mosek.stakey[] skx)
```

Obtains the status keys for the scalar variables.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `skx` (*mosek.stakey*[]) – Status keys for the variables. (output)

Groups *Solution (get)*

Task.getskxslice

```
void getskxslice
(mosek.soltype whichsol,
 int first,
 int last,
 mosek.stakey[] skx)
```

Obtains the status keys for a slice of the scalar variables.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (int) – First index in the sequence. (input)
- `last` (int) – Last index plus 1 in the sequence. (input)
- `skx` (*mosek.stakey*[]) – Status keys for the variables. (output)

Groups *Solution (get)*

Task.getslc

```
void getslc
(mosek.soltype whichsol,
 double[] slc)
```

Obtains the s_l^c vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `slc` (double[]) – Dual variables corresponding to the lower bounds on the constraints. (output)

Groups *Solution (get)*

Task.getslcslice

```
void getslcslice
(mosek.soltype whichsol,
 int first,
 int last,
 double[] slc)
```

Obtains a slice of the s_l^c vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)

- `first (int)` – First index in the sequence. (input)
- `last (int)` – Last index plus 1 in the sequence. (input)
- `slc (double[])` – Dual variables corresponding to the lower bounds on the constraints. (output)

Groups *Solution (get)*

`Task.getslx`

```
void getslx
(mosek.soltype whichsol,
 double[] slx)
```

Obtains the s_l^x vector for a solution.

Parameters

- `whichsol (mosek.soltype)` – Selects a solution. (input)
- `slx (double[])` – Dual variables corresponding to the lower bounds on the variables. (output)

Groups *Solution (get)*

`Task.getslxslice`

```
void getslxslice
(mosek.soltype whichsol,
 int first,
 int last,
 double[] slx)
```

Obtains a slice of the s_l^x vector for a solution.

Parameters

- `whichsol (mosek.soltype)` – Selects a solution. (input)
- `first (int)` – First index in the sequence. (input)
- `last (int)` – Last index plus 1 in the sequence. (input)
- `slx (double[])` – Dual variables corresponding to the lower bounds on the variables. (output)

Groups *Solution (get)*

`Task.getsnx`

```
void getsnx
(mosek.soltype whichsol,
 double[] snx)
```

Obtains the s_n^x vector for a solution.

Parameters

- `whichsol (mosek.soltype)` – Selects a solution. (input)
- `snx (double[])` – Dual variables corresponding to the conic constraints on the variables. (output)

Groups *Solution (get)*

Task.getsnxslice

```
void getsnxslice
(mosek.soltype whichsol,
 int first,
 int last,
 double[] snx)
```

Obtains a slice of the s_n^x vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (int) – First index in the sequence. (input)
- `last` (int) – Last index plus 1 in the sequence. (input)
- `snx` (double[]) – Dual variables corresponding to the conic constraints on the variables. (output)

Groups *Solution (get)*

Task.getsolsta

```
void getsolsta
(mosek.soltype whichsol,
 mosek.solsta[] solsta)
```

```
mosek.solsta getsolsta (mosek.soltype whichsol)
```

Obtains the solution status.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `solsta` (*mosek.solsta by reference*) – Solution status. (output)

Return (*mosek.solsta*) – Solution status.

Groups *Solution information*

Task.getsolution

```
void getsolution
(mosek.soltype whichsol,
 mosek.prosta[] prosta,
 mosek.solsta[] solsta,
 mosek.stakey[] skc,
 mosek.stakey[] skx,
 mosek.stakey[] skn,
 double[] xc,
 double[] xx,
 double[] y,
 double[] slc,
 double[] suc,
 double[] slx,
 double[] sux,
 double[] snx)
```

Obtains the complete solution.

Consider the case of linear programming. The primal problem is given by

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

and the corresponding dual problem is

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

A conic optimization problem has the same primal variables as in the linear case. Recall that the dual of a conic optimization problem is given by:

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

The mapping between variables and arguments to the function is as follows:

- **xx** : Corresponds to variable x (also denoted x^x).
- **xc** : Corresponds to $x^c := Ax$.
- **y** : Corresponds to variable y .
- **slc** : Corresponds to variable s_l^c .
- **suc** : Corresponds to variable s_u^c .
- **slx** : Corresponds to variable s_l^x .
- **sux** : Corresponds to variable s_u^x .
- **snx** : Corresponds to variable s_n^x .

The meaning of the values returned by this function depend on the *solution status* returned in the argument **solsta**. The most important possible values of **solsta** are:

- **solsta.optimal** : An optimal solution satisfying the optimality criteria for continuous problems is returned.
- **solsta.integer_optimal** : An optimal solution satisfying the optimality criteria for integer problems is returned.
- **solsta.prim_feas** : A solution satisfying the feasibility criteria.
- **solsta.prim_infeas_cer** : A primal certificate of infeasibility is returned.
- **solsta.dual_infeas_cer** : A dual certificate of infeasibility is returned.

In order to retrieve the primal and dual values of semidefinite variables see [Task.getbarxj](#) and [Task.getbarsj](#).

Parameters

- **whichsol** ([mosek.soltype](#)) – Selects a solution. (input)
- **prosta** ([mosek.prosta by reference](#)) – Problem status. (output)
- **solsta** ([mosek.solsta by reference](#)) – Solution status. (output)
- **skc** ([mosek.stakey \[\]](#)) – Status keys for the constraints. (output)

- `skx` (*mosek.stakey*[]) – Status keys for the variables. (output)
- `skn` (*mosek.stakey*[]) – Status keys for the conic constraints. (output)
- `xc` (double[]) – Primal constraint solution. (output)
- `xx` (double[]) – Primal variable solution. (output)
- `y` (double[]) – Vector of dual variables corresponding to the constraints. (output)
- `slc` (double[]) – Dual variables corresponding to the lower bounds on the constraints. (output)
- `suc` (double[]) – Dual variables corresponding to the upper bounds on the constraints. (output)
- `slx` (double[]) – Dual variables corresponding to the lower bounds on the variables. (output)
- `sux` (double[]) – Dual variables corresponding to the upper bounds on the variables. (output)
- `snx` (double[]) – Dual variables corresponding to the conic constraints on the variables. (output)

Groups *Solution (get)*

`Task.getsolutioni`

```
void getsolutioni
(mosek.accmode accmode,
 int i,
 mosek.soltype whichsol,
 mosek.stakey[] sk,
 double[] x,
 double[] sl,
 double[] su,
 double[] sn)
```

Obtains the primal and dual solution information for a single constraint or variable.

Parameters

- `accmode` (*mosek.accmode*) – Defines whether solution information for a constraint or for a variable is retrieved. (input)
- `i` (int) – Index of the constraint or variable. (input)
- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `sk` (*mosek.stakey by reference*) – Status key of the constraint of variable. (output)
- `x` (double *by reference*) – Solution value of the primal variable. (output)
- `sl` (double *by reference*) – Solution value of the dual variable associated with the lower bound. (output)
- `su` (double *by reference*) – Solution value of the dual variable associated with the upper bound. (output)
- `sn` (double *by reference*) – Solution value of the dual variable associated with the cone constraint. (output)

Groups *Solution (get)*

`Task.getsolutioninfo`

```

void getsolutioninfo
(mosek.soltype whichsol,
 double[] pobj,
 double[] pviolcon,
 double[] pviolvar,
 double[] pviolbarvar,
 double[] pviolcone,
 double[] pviolitg,
 double[] dobj,
 double[] dviolcon,
 double[] dviolvar,
 double[] dviolbarvar,
 double[] dviolcone)

```

Obtains information about a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `pobj` (double *by reference*) – The primal objective value as computed by *Task.getprimalobj*. (output)
- `pviolcon` (double *by reference*) – Maximal primal violation of the solution associated with the x^c variables where the violations are computed by *Task.getpviolcon*. (output)
- `pviolvar` (double *by reference*) – Maximal primal violation of the solution for the x variables where the violations are computed by *Task.getpviolvar*. (output)
- `pviolbarvar` (double *by reference*) – Maximal primal violation of solution for the \bar{X} variables where the violations are computed by *Task.getpviolbarvar*. (output)
- `pviolcone` (double *by reference*) – Maximal primal violation of solution for the conic constraints where the violations are computed by *Task.getpviolcones*. (output)
- `pviolitg` (double *by reference*) – Maximal violation in the integer constraints. The violation for an integer variable x_j is given by $\min(x_j - \lfloor x_j \rfloor, \lceil x_j \rceil - x_j)$. This number is always zero for the interior-point and basic solutions. (output)
- `dobj` (double *by reference*) – Dual objective value as computed by *Task.getdualobj*. (output)
- `dviolcon` (double *by reference*) – Maximal violation of the dual solution associated with the x^c variable as computed by *Task.getdviolcon*. (output)
- `dviolvar` (double *by reference*) – Maximal violation of the dual solution associated with the x variable as computed by *Task.getdviolvar*. (output)
- `dviolbarvar` (double *by reference*) – Maximal violation of the dual solution associated with the \bar{S} variable as computed by *Task.getdviolbarvar*. (output)
- `dviolcone` (double *by reference*) – Maximal violation of the dual solution associated with the dual conic constraints as computed by *Task.getdviolcones*. (output)

Groups *Solution information*

`Task.getsolutionslice`

```

void getsolutionslice
(mosek.soltype whichsol,

```

```
mosek.solitem solitem,
int first,
int last,
double[] values)
```

Obtains a slice of one item from the solution. The format of the solution is exactly as in *Task.getsolution*. The parameter *solitem* determines which of the solution vectors should be returned.

Parameters

- *whichsol* (*mosek.soltype*) – Selects a solution. (input)
- *solitem* (*mosek.solitem*) – Which part of the solution is required. (input)
- *first* (int) – First index in the sequence. (input)
- *last* (int) – Last index plus 1 in the sequence. (input)
- *values* (double[]) – The values in the required sequence are stored sequentially in *values*. (output)

Groups *Solution (get)*

Task.getsparsesymmat

```
void getsparsesymmat
(long idx,
int[] subi,
int[] subj,
double[] valij)
```

Get a single symmetric matrix from the matrix store.

Parameters

- *idx* (long) – Index of the matrix to retrieve. (input)
- *subi* (int[]) – Row subscripts of the matrix non-zero elements. (output)
- *subj* (int[]) – Column subscripts of the matrix non-zero elements. (output)
- *valij* (double[]) – Coefficients of the matrix non-zero elements. (output)

Groups *Scalar variable data*

Task.getstrparam

```
void getstrparam
(mosek.sparam param,
int[] len,
StringBuffer parvalue)
```

```
String getstrparam
(mosek.sparam param,
int[] len)
```

Obtains the value of a string parameter.

Parameters

- *param* (*mosek.sparam*) – Which parameter. (input)
- *len* (int *by reference*) – The length of the parameter value. (output)
- *parvalue* (StringBuffer) – Parameter value. (output)

Return (String) – Parameter value.

Groups *Parameters (get)*

Task.getstrparamlen

```
void getstrparamlen
(mosek.sparam param,
 int[] len)
```

```
int getstrparamlen (mosek.sparam param)
```

Obtains the length of a string parameter.

Parameters

- param (*mosek.sparam*) – Which parameter. (input)
- len (int *by reference*) – The length of the parameter value. (output)

Return (int) – The length of the parameter value.

Groups *Parameters (get)*

Task.getsuc

```
void getsuc
(mosek.soltype whichsol,
 double[] suc)
```

Obtains the s_u^c vector for a solution.

Parameters

- whichsol (*mosek.soltype*) – Selects a solution. (input)
- suc (double[]) – Dual variables corresponding to the upper bounds on the constraints. (output)

Groups *Solution (get)*

Task.getsucslice

```
void getsucslice
(mosek.soltype whichsol,
 int first,
 int last,
 double[] suc)
```

Obtains a slice of the s_u^c vector for a solution.

Parameters

- whichsol (*mosek.soltype*) – Selects a solution. (input)
- first (int) – First index in the sequence. (input)
- last (int) – Last index plus 1 in the sequence. (input)
- suc (double[]) – Dual variables corresponding to the upper bounds on the constraints. (output)

Groups *Solution (get)*

Task.getsux


```
void getsux
(mosek.soltype whichsol,
 double[] sux)
```

Obtains the s_u^x vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `sux` (`double[]`) – Dual variables corresponding to the upper bounds on the variables. (output)

Groups *Solution (get)*

`Task.getsuxslice`

```
void getsuxslice
(mosek.soltype whichsol,
 int first,
 int last,
 double[] sux)
```

Obtains a slice of the s_u^x vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (`int`) – First index in the sequence. (input)
- `last` (`int`) – Last index plus 1 in the sequence. (input)
- `sux` (`double[]`) – Dual variables corresponding to the upper bounds on the variables. (output)

Groups *Solution (get)*

`Task.getsymmatinfo`

```
void getsymmatinfo
(long idx,
 int[] dim,
 long[] nz,
 mosek.symmattype[] type)
```

MOSEK maintains a vector denoted by E of symmetric data matrices. This function makes it possible to obtain important information about a single matrix in E .

Parameters

- `idx` (`long`) – Index of the matrix for which information is requested. (input)
- `dim` (`int` *by reference*) – Returns the dimension of the requested matrix. (output)
- `nz` (`long` *by reference*) – Returns the number of non-zeros in the requested matrix. (output)
- `type` (*mosek.symmattype by reference*) – Returns the type of the requested matrix. (output)

Groups *Scalar variable data*

`Task.gettaskname`

```
void gettaskname (StringBuffer taskname)
```

```
String gettaskname ()
```

Obtains the name assigned to the task.

Parameters `taskname (StringBuffer)` – Returns the task name. (output)

Return (`String`) – Returns the task name.

Groups *Naming*

`Task.gettasknamelen`

```
void gettasknamelen (int[] len)
```

```
int gettasknamelen ()
```

Obtains the length the task name.

Parameters `len (int by reference)` – Returns the length of the task name. (output)

Return (`int`) – Returns the length of the task name.

Groups *Naming*

`Task.getvarbound`

```
void getvarbound  
(int i,  
 mosek.boundkey[] bk,  
 double[] bl,  
 double[] bu)
```

Obtains bound information for one variable.

Parameters

- `i (int)` – Index of the variable for which the bound information should be obtained. (input)
- `bk (mosek.boundkey by reference)` – Bound keys. (output)
- `bl (double by reference)` – Values for lower bounds. (output)
- `bu (double by reference)` – Values for upper bounds. (output)

Groups *Bound data*

`Task.getvarboundslice`

```
void getvarboundslice  
(int first,  
 int last,  
 mosek.boundkey[] bk,  
 double[] bl,  
 double[] bu)
```

Obtains bounds information for a slice of the variables.

Parameters

- `first (int)` – First index in the sequence. (input)
- `last (int)` – Last index plus 1 in the sequence. (input)

- `bk` (*mosek.boundkey* []) – Bound keys. (output)
- `bl` (`double` []) – Values for lower bounds. (output)
- `bu` (`double` []) – Values for upper bounds. (output)

Groups *Bound data*

`Task.getvarname`

```
void getvarname
(int j,
 StringBuffer name)
```

```
String getvarname (int j)
```

Obtains the name of a variable.

Parameters

- `j` (`int`) – Index of a variable. (input)
- `name` (`StringBuffer`) – Returns the required name. (output)

Return (`String`) – Returns the required name.

Groups *Naming*

`Task.getvarnameindex`

```
void getvarnameindex
(String somename,
 int[] asgn,
 int[] index)
```

```
int getvarnameindex
(String somename,
 int[] asgn)
```

Checks whether the name `somename` has been assigned to any variable. If so, the index of the variable is reported.

Parameters

- `somename` (`String`) – The name which should be checked. (input)
- `asgn` (`int` *by reference*) – Is non-zero if the name `somename` is assigned to a variable. (output)
- `index` (`int` *by reference*) – If the name `somename` is assigned to a variable, then `index` is the index of the variable. (output)

Return (`int`) – If the name `somename` is assigned to a variable, then `index` is the index of the variable.

Groups *Naming*

`Task.getvarnamelen`

```
void getvarnamelen
(int i,
 int[] len)
```

```
int getvarnamelen (int i)
```

Obtains the length of the name of a variable.

Parameters

- `i` (`int`) – Index of a variable. (input)
- `len` (`int` *by reference*) – Returns the length of the indicated name. (output)

Return (`int`) – Returns the length of the indicated name.

Groups *Naming*

`Task.getvartype`

```
void getvartype  
(int j,  
 mosek.variabletype[] vartype)
```

```
mosek.variabletype getvartype (int j)
```

Gets the variable type of one variable.

Parameters

- `j` (`int`) – Index of the variable. (input)
- `vartype` (*`mosek.variabletype` by reference*) – Variable type of the j -th variable. (output)

Return (*`mosek.variabletype`*) – Variable type of the j -th variable.

Groups *Scalar variable data*

`Task.getvartypelist`

```
void getvartypelist  
(int[] subj,  
 mosek.variabletype[] vartype)
```

Obtains the variable type of one or more variables. Upon return `vartype[k]` is the variable type of variable `subj[k]`.

Parameters

- `subj` (`int[]`) – A list of variable indexes. (input)
- `vartype` (*`mosek.variabletype[]`*) – The variables types corresponding to the variables specified by `subj`. (output)

Groups *Scalar variable data*

`Task.getxc`

```
void getxc  
(mosek.soltype whichsol,  
 double[] xc)
```

Obtains the x^c vector for a solution.

Parameters

- `whichsol` (*`mosek.soltype`*) – Selects a solution. (input)
- `xc` (`double[]`) – Primal constraint solution. (output)

Groups *Solution (get)*

Task.getxcslice

```
void getxcslice
(mosek.soltype whichsol,
 int first,
 int last,
 double[] xc)
```

Obtains a slice of the x^c vector for a solution.

Parameters

- **whichsol** (*mosek.soltype*) – Selects a solution. (input)
- **first** (int) – First index in the sequence. (input)
- **last** (int) – Last index plus 1 in the sequence. (input)
- **xc** (double[]) – Primal constraint solution. (output)

Groups *Solution (get)*

Task.getxx

```
void getxx
(mosek.soltype whichsol,
 double[] xx)
```

Obtains the x^x vector for a solution.

Parameters

- **whichsol** (*mosek.soltype*) – Selects a solution. (input)
- **xx** (double[]) – Primal variable solution. (output)

Groups *Solution (get)*

Task.getxxslice

```
void getxxslice
(mosek.soltype whichsol,
 int first,
 int last,
 double[] xx)
```

Obtains a slice of the x^x vector for a solution.

Parameters

- **whichsol** (*mosek.soltype*) – Selects a solution. (input)
- **first** (int) – First index in the sequence. (input)
- **last** (int) – Last index plus 1 in the sequence. (input)
- **xx** (double[]) – Primal variable solution. (output)

Groups *Solution (get)*

Task.gety

```
void gety
(mosek.soltype whichsol,
 double[] y)
```

Obtains the y vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `y` (`double[]`) – Vector of dual variables corresponding to the constraints. (output)

Groups *Solution (get)*

`Task.getyslice`

```
void getyslice
(mosek.soltype whichsol,
 int first,
 int last,
 double[] y)
```

Obtains a slice of the y vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (`int`) – First index in the sequence. (input)
- `last` (`int`) – Last index plus 1 in the sequence. (input)
- `y` (`double[]`) – Vector of dual variables corresponding to the constraints. (output)

Groups *Solution (get)*

`Task.initbasissolve`

```
void initbasissolve (int[] basis)
```

Prepare a task for use with the *Task.solvewithbasis* function.

This function should be called

- immediately before the first call to *Task.solvewithbasis*, and
- immediately before any subsequent call to *Task.solvewithbasis* if the task has been modified.

If the basis is singular i.e. not invertible, then the error *rescode.err_basis_singular* is reported.

Parameters `basis` (`int[]`) – The array of basis indexes to use. The array is interpreted as follows: If `basis[i] ≤ numcon − 1`, then $x_{\text{basis}[i]}^c$ is in the basis at position i , otherwise $x_{\text{basis}[i] - \text{numcon}}$ is in the basis at position i . (output)

Groups *Basis matrix*

`Task.inputdata`

```
void inputdata
(int maxnumcon,
 int maxnumvar,
 double[] c,
```

```

double cfix,
int[] aptrb,
int[] aptre,
int[] asub,
double[] aval,
mosek.boundkey[] bkc,
double[] blc,
double[] buc,
mosek.boundkey[] bkc,
double[] blx,
double[] bux)

```

```

void inputdata
(int maxnumcon,
int maxnumvar,
double[] c,
double cfix,
long[] aptrb,
long[] aptre,
int[] asub,
double[] aval,
mosek.boundkey[] bkc,
double[] blc,
double[] buc,
mosek.boundkey[] bkc,
double[] blx,
double[] bux)

```

Input the linear part of an optimization task in one function call.

Parameters

- `maxnumcon` (int) – Number of preallocated constraints in the optimization task. (input)
- `maxnumvar` (int) – Number of preallocated variables in the optimization task. (input)
- `c` (double[]) – Linear terms of the objective as a dense vector. The length is the number of variables. (input)
- `cfix` (double) – Fixed term in the objective. (input)
- `aptrb` (int[]) – Row or column start pointers. (input)
- `aptrb` (long[]) – Row or column start pointers. (input)
- `aptre` (int[]) – Row or column end pointers. (input)
- `aptre` (long[]) – Row or column end pointers. (input)
- `asub` (int[]) – Coefficient subscripts. (input)
- `aval` (double[]) – Coefficient values. (input)
- `bkc` (*mosek.boundkey*[]) – Bound keys for the constraints. (input)
- `blc` (double[]) – Lower bounds for the constraints. (input)
- `buc` (double[]) – Upper bounds for the constraints. (input)
- `bkc` (*mosek.boundkey*[]) – Bound keys for the variables. (input)
- `blx` (double[]) – Lower bounds for the variables. (input)
- `bux` (double[]) – Upper bounds for the variables. (input)

Groups *Task management*

Task.isdoupurname

```
void isdoupurname
(String purname,
 mosek.dparam[] param)
```

Checks whether purname is a valid double parameter name.

Parameters

- purname (String) – Parameter name. (input)
- param (*mosek.dparam by reference*) – Returns the parameter corresponding to the name, if one exists. (output)

Groups *Parameter management*

Task.isintpname

```
void isintpname
(String purname,
 mosek.iparam[] param)
```

Checks whether purname is a valid integer parameter name.

Parameters

- purname (String) – Parameter name. (input)
- param (*mosek.iparam by reference*) – Returns the parameter corresponding to the name, if one exists. (output)

Groups *Parameter management*

Task.isstrpname

```
void isstrpname
(String purname,
 mosek.sparam[] param)
```

Checks whether purname is a valid string parameter name.

Parameters

- purname (String) – Parameter name. (input)
- param (*mosek.sparam by reference*) – Returns the parameter corresponding to the name, if one exists. (output)

Groups *Parameter management*

Task.linkfiletostream

```
void linkfiletostream
(mosek.streamtype whichstream,
 String filename,
 int append)
```

Directs all output from a task stream `whichstream` to a file `filename`.

Parameters

- whichstream (*mosek.streamtype*) – Index of the stream. (input)
- filename (String) – A valid file name. (input)

- `append (int)` – If this argument is 0 the output file will be overwritten, otherwise it will be appended to. (input)

Groups *Logging*

Task.onesolutionsummary

```
void onesolutionsummary
(mosek.streamtype whichstream,
 mosek.soltype whichsol)
```

Prints a short summary of a specified solution.

Parameters

- `whichstream (mosek.streamtype)` – Index of the stream. (input)
- `whichsol (mosek.soltype)` – Selects a solution. (input)

Groups *Task diagnostics*

Task.optimize

```
void optimize (mosek.rescode[] trmcode)
```

```
mosek.rescode 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*.

Parameters `trmcode (mosek.rescode by reference)` – Is either *rescode.ok* or a termination response code. (output)

Return `(mosek.rescode)` – Is either *rescode.ok* or a termination response code.

Groups *Optimization*

Task.optimizermt

```
void optimizermt
(String server,
 String port,
 mosek.rescode[] trmcode)
```

Offload the optimization task to a solver server defined by `server:port`. The call will block until a result is available or the connection closes.

If the string parameter *sparam.remote_access_token* is not blank, it will be passed to the server as authentication.

Parameters

- `server (String)` – Name or IP address of the solver server. (input)
- `port (String)` – Network port of the solver server. (input)
- `trmcode (mosek.rescode by reference)` – Is either *rescode.ok* or a termination response code. (output)

Task.optimizersummary

```
void optimizersummary (mosek.streamtype whichstream)
```

Prints a short summary with optimizer statistics from last optimization.

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

Groups *Task diagnostics*

`Task.primalrepair`

```
void primalrepair
(double[] wlc,
 double[] wuc,
 double[] wlx,
 double[] wux)
```

The function repairs a primal infeasible optimization problem by adjusting the bounds on the constraints and variables where the adjustment is computed as the minimal weighted sum of relaxations to the bounds on the constraints and variables. Observe the function only repairs the problem but does not solve it. If an optimal solution is required the problem should be optimized after the repair.

The function is applicable to linear and conic problems possibly with integer variables.

Observe that when computing the minimal weighted relaxation the termination tolerance specified by the parameters of the task is employed. For instance the parameter *iparam.mio_mode* can be used to make **MOSEK** ignore the integer constraints during the repair which usually leads to a much faster repair. However, the drawback is of course that the repaired problem may not have an integer feasible solution.

Note the function modifies the task in place. If this is not desired, then apply the function to a cloned task.

Parameters

- `wlc` (`double[]`) – $(w_l^c)_i$ is the weight associated with relaxing the lower bound on constraint i . If the weight is negative, then the lower bound is not relaxed. Moreover, if the argument is NULL, then all the weights are assumed to be 1. (input)
- `wuc` (`double[]`) – $(w_u^c)_i$ is the weight associated with relaxing the upper bound on constraint i . If the weight is negative, then the upper bound is not relaxed. Moreover, if the argument is NULL, then all the weights are assumed to be 1. (input)
- `wlx` (`double[]`) – $(w_l^x)_j$ is the weight associated with relaxing the lower bound on variable j . If the weight is negative, then the lower bound is not relaxed. Moreover, if the argument is NULL, then all the weights are assumed to be 1. (input)
- `wux` (`double[]`) – $(w_u^x)_j$ is the weight associated with relaxing the upper bound on variable j . If the weight is negative, then the upper bound is not relaxed. Moreover, if the argument is NULL, then all the weights are assumed to be 1. (input)

Groups *Infeasibility diagnostics*

`Task.primalsensitivity`

```
void primalsensitivity
(int[] subi,
 mosek.mark[] marki,
 int[] subj,
 mosek.mark[] markj,
 double[] leftpricei,
 double[] rightpricei,
```

```
double[] leftrangei,
double[] rightrangei,
double[] leftpricej,
double[] rightpricej,
double[] leftrangej,
double[] rightrangej)
```

Calculates sensitivity information for bounds on variables and constraints. For details on sensitivity analysis, the definitions of *shadow price* and *linearity interval* and an example see Section 17.

The type of sensitivity analysis to be performed (basis or optimal partition) is controlled by the parameter *iparam.sensitivity_type*.

Parameters

- **subi** (int[]) – Indexes of constraints to analyze. (input)
- **marki** (*mosek.mark*[]) – The value of **marki**[i] indicates for which bound of constraint **subi**[i] sensitivity analysis is performed. If **marki**[i] = *mark.up* the upper bound of constraint **subi**[i] is analyzed, and if **marki**[i] = *mark.lo* the lower bound is analyzed. If **subi**[i] is an equality constraint, either *mark.lo* or *mark.up* can be used to select the constraint for sensitivity analysis. (input)
- **subj** (int[]) – Indexes of variables to analyze. (input)
- **markj** (*mosek.mark*[]) – The value of **markj**[j] indicates for which bound of variable **subj**[j] sensitivity analysis is performed. If **markj**[j] = *mark.up* the upper bound of variable **subj**[j] is analyzed, and if **markj**[j] = *mark.lo* the lower bound is analyzed. If **subj**[j] is a fixed variable, either *mark.lo* or *mark.up* can be used to select the bound for sensitivity analysis. (input)
- **leftpricei** (double[]) – **leftpricei**[i] is the left shadow price for the bound **marki**[i] of constraint **subi**[i]. (output)
- **rightpricei** (double[]) – **rightpricei**[i] is the right shadow price for the bound **marki**[i] of constraint **subi**[i]. (output)
- **leftrangei** (double[]) – **leftrangei**[i] is the left range β_1 for the bound **marki**[i] of constraint **subi**[i]. (output)
- **rightrangei** (double[]) – **rightrangei**[i] is the right range β_2 for the bound **marki**[i] of constraint **subi**[i]. (output)
- **leftpricej** (double[]) – **leftpricej**[j] is the left shadow price for the bound **markj**[j] of variable **subj**[j]. (output)
- **rightpricej** (double[]) – **rightpricej**[j] is the right shadow price for the bound **markj**[j] of variable **subj**[j]. (output)
- **leftrangej** (double[]) – **leftrangej**[j] is the left range β_1 for the bound **markj**[j] of variable **subj**[j]. (output)
- **rightrangej** (double[]) – **rightrangej**[j] is the right range β_2 for the bound **markj**[j] of variable **subj**[j]. (output)

Groups *Sensitivity analysis*

Task.putacol

```
void putacol
(int j,
 int[] subj,
 double[] valj)
```

Change one column of the linear constraint matrix A . Resets all the elements in column j to zero and then sets

$$a_{\text{subj}[k],j} = \text{valj}[k], \quad k = 0, \dots, \text{nzj} - 1.$$

Parameters

- j (int) – Index of a column in A . (input)
- subj (int[]) – Row indexes of non-zero values in column j of A . (input)
- valj (double[]) – New non-zero values of column j in A . (input)

Groups *Scalar variable data*

Task.putacollist

```
void putacollist
(int[] sub,
 int[] ptrb,
 int[] ptre,
 int[] asub,
 double[] aval)
```

Change a set of columns in the linear constraint matrix A with data in sparse triplet format. The requested columns are set to zero and then updated with:

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

Parameters

- sub (int[]) – Indexes of columns that should be replaced, no duplicates. (input)
- ptrb (int[]) – Array of pointers to the first element in each column. (input)
- ptre (int[]) – Array of pointers to the last element plus one in each column. (input)
- asub (int[]) – Row indexes of new elements. (input)
- aval (double[]) – Coefficient values. (input)

Groups *Scalar variable data*

Task.putacolslice

```
void putacolslice
(int first,
 int last,
 int[] ptrb,
 int[] ptre,
 int[] asub,
 double[] aval)
```

```
void putacolslice
(int first,
 int last,
 long[] ptrb,
 long[] ptre,
 int[] asub,
 double[] aval)
```

Change a slice of columns in the linear constraint matrix A with data in sparse triplet format. The requested columns are set to zero and then updated with:

$$\text{for } i = \text{first}, \dots, \text{last} - 1 \\ a_{\text{asub}[k], i} = \text{aval}[k], \quad k = \text{ptrb}[i], \dots, \text{ptre}[i] - 1.$$

Parameters

- **first** (int) – First column in the slice. (input)
- **last** (int) – Last column plus one in the slice. (input)
- **ptrb** (int[]) – Array of pointers to the first element in each column. (input)
- **ptrb** (long[]) – Array of pointers to the first element in each column. (input)
- **ptre** (int[]) – Array of pointers to the last element plus one in each column. (input)
- **ptre** (long[]) – Array of pointers to the last element plus one in each column. (input)
- **asub** (int[]) – Row indexes of new elements. (input)
- **aval** (double[]) – Coefficient values. (input)

Groups *Scalar variable data*

Task.putaij

```
void putaij
(int i,
 int j,
 double aij)
```

Changes a coefficient in the linear coefficient matrix A using the method

$$a_{i,j} = \text{aij}.$$

Parameters

- **i** (int) – Constraint (row) index. (input)
- **j** (int) – Variable (column) index. (input)
- **aij** (double) – New coefficient for $a_{i,j}$. (input)

Groups *Scalar variable data*

Task.putaijlist

```
void putaijlist
(int[] subi,
 int[] subj,
 double[] valij)
```

Changes one or more coefficients in A using the method

$$a_{\text{subi}[k], \text{subj}[k]} = \text{valij}[k], \quad k = 0, \dots, \text{num} - 1.$$

Duplicates are not allowed.

Parameters

- **subi** (int[]) – Constraint (row) indices. (input)
- **subj** (int[]) – Variable (column) indices. (input)

- `valij (double[])` – New coefficient values for $a_{i,j}$. (input)

Groups *Scalar variable data*

`Task.putarow`

```
void putarow
(int i,
 int[] subi,
 double[] vali)
```

Change one column of the linear constraint matrix A . Resets all the elements in row i to zero and then sets

$$a_{i,\text{subi}[k]} = \text{vali}[k], \quad k = 0, \dots, \text{nzi} - 1.$$

Parameters

- `i (int)` – Index of a row in A . (input)
- `subi (int[])` – Column indexes of non-zero values in row i of A . (input)
- `vali (double[])` – New non-zero values of row i in A . (input)

Groups *Scalar variable data*

`Task.putarowlist`

```
void putarowlist
(int[] sub,
 int[] ptrb,
 int[] ptre,
 int[] asub,
 double[] aval)
```

Change a set of rows in the linear constraint matrix A with data in sparse triplet format. The requested rows are set to zero and then updated with:

$$\begin{aligned} \text{for } i = 0, \dots, \text{num} - 1 \\ a_{\text{sub}[i], \text{asub}[k]} = \text{aval}[k], \quad k = \text{ptrb}[i], \dots, \text{ptre}[i] - 1. \end{aligned}$$

Parameters

- `sub (int[])` – Indexes of rows that should be replaced, no duplicates. (input)
- `ptrb (int[])` – Array of pointers to the first element in each row. (input)
- `ptre (int[])` – Array of pointers to the last element plus one in each row. (input)
- `asub (int[])` – Column indexes of new elements. (input)
- `aval (double[])` – Coefficient values. (input)

Groups *Scalar variable data*

`Task.putarowslice`

```
void putarowslice
(int first,
 int last,
 int[] ptrb,
 int[] ptre,
 int[] asub,
 double[] aval)
```

```

void putarowslice
(int first,
 int last,
 long[] ptrb,
 long[] ptre,
 int[] asub,
 double[] aval)

```

Change a slice of rows in the linear constraint matrix A with data in sparse triplet format. The requested columns are set to zero and then updated with:

$$\text{for } i = \text{first}, \dots, \text{last} - 1 \\ a_{\text{sub}[i], \text{asub}[k]} = \text{aval}[k], \quad k = \text{ptrb}[i], \dots, \text{ptre}[i] - 1.$$

Parameters

- **first** (int) – First row in the slice. (input)
- **last** (int) – Last row plus one in the slice. (input)
- **ptrb** (int[]) – Array of pointers to the first element in each row. (input)
- **ptrb** (long[]) – Array of pointers to the first element in each row. (input)
- **ptre** (int[]) – Array of pointers to the last element plus one in each row. (input)
- **ptre** (long[]) – Array of pointers to the last element plus one in each row. (input)
- **asub** (int[]) – Column indexes of new elements. (input)
- **aval** (double[]) – Coefficient values. (input)

Groups *Scalar variable data*

Task.putbarablocktriplet

```

void putbarablocktriplet
(long num,
 int[] subi,
 int[] subj,
 int[] subk,
 int[] subl,
 double[] valijkl)

```

Inputs the \bar{A} matrix in block triplet form.

Parameters

- **num** (long) – Number of elements in the block triplet form. (input)
- **subi** (int[]) – Constraint index. (input)
- **subj** (int[]) – Symmetric matrix variable index. (input)
- **subk** (int[]) – Block row index. (input)
- **subl** (int[]) – Block column index. (input)
- **valijkl** (double[]) – The numerical value associated with each block triplet. (input)

Groups *Symmetric matrix variable data*

Task.putbaraij

```
void putbaraij
(int i,
 int j,
 long[] sub,
 double[] weights)
```

This function sets one element in the \bar{A} matrix.

Each element in the \bar{A} matrix is a weighted sum of symmetric matrices from the symmetric matrix storage E , so \bar{A}_{ij} is a symmetric matrix. By default all elements in \bar{A} are 0, so only non-zero elements need be added. Setting the same element again will overwrite the earlier entry.

The symmetric matrices from E are defined separately using the function *Task.appendsparsesymmat*.

Parameters

- **i** (int) – Row index of \bar{A} . (input)
- **j** (int) – Column index of \bar{A} . (input)
- **sub** (long[]) – Indices in E of the matrices appearing in the weighted sum for \bar{A}_{ij} . (input)
- **weights** (double[]) – **weights[k]** is the coefficient of the **sub[k]**-th element of E in the weighted sum forming \bar{A}_{ij} . (input)

Groups *Symmetric matrix variable data*

Task.putbarblocktriplet

```
void putbarblocktriplet
(long num,
 int[] subj,
 int[] subk,
 int[] subl,
 double[] valjkl)
```

Inputs the \bar{C} matrix in block triplet form.

Parameters

- **num** (long) – Number of elements in the block triplet form. (input)
- **subj** (int[]) – Symmetric matrix variable index. (input)
- **subk** (int[]) – Block row index. (input)
- **subl** (int[]) – Block column index. (input)
- **valjkl** (double[]) – The numerical value associated with each block triplet. (input)

Groups *Symmetric matrix variable data*

Task.putbarcj

```
void putbarcj
(int j,
 long[] sub,
 double[] weights)
```

This function sets one entry in the \bar{C} vector.

Each element in the \bar{C} vector is a weighted sum of symmetric matrices from the symmetric matrix storage E , so \bar{C}_j is a symmetric matrix. By default all elements in \bar{C} are 0, so only non-zero elements need be added. Setting the same element again will overwrite the earlier entry.

The symmetric matrices from E are defined separately using the function [Task.appendsparsesymmat](#).

Parameters

- `j` (`int`) – Index of the element in \bar{C} that should be changed. (input)
- `sub` (`long[]`) – Indices in E of matrices appearing in the weighted sum for \bar{C}_j (input)
- `weights` (`double[]`) – `weights[k]` is the coefficient of the `sub[k]`-th element of E in the weighted sum forming \bar{C}_j . (input)

Groups [Symmetric matrix variable data](#)

`Task.putbarsj`

```
void putbarsj
(mosek.soltype whichsol,
 int j,
 double[] barsj)
```

Sets the dual solution for a semidefinite variable.

Parameters

- `whichsol` ([mosek.soltype](#)) – Selects a solution. (input)
- `j` (`int`) – Index of the semidefinite variable. (input)
- `barsj` (`double[]`) – Value of \bar{S}_j . Format as in [Task.getbarsj](#). (input)

Groups [Solution \(put\)](#)

`Task.putbarvarname`

```
void putbarvarname
(int j,
 String name)
```

Sets the name of a semidefinite variable.

Parameters

- `j` (`int`) – Index of the variable. (input)
- `name` (`String`) – The variable name. (input)

Groups [Naming](#)

`Task.putbarxj`

```
void putbarxj
(mosek.soltype whichsol,
 int j,
 double[] barxj)
```

Sets the primal solution for a semidefinite variable.

Parameters

- `whichsol` ([mosek.soltype](#)) – Selects a solution. (input)

- `j` (`int`) – Index of the semidefinite variable. (input)
- `barxj` (`double[]`) – Value of \bar{X}_j . Format as in *Task.getbarxj*. (input)

Groups *Solution (put)*

`Task.putbound`

```
void putbound
(mosek.acemode accmode,
 int i,
 mosek.boundkey bk,
 double bl,
 double bu)
```

Changes the bound for either one constraint or one variable.

If the bound value specified is numerically larger than *dparam.data_tol_bound_inf* it is considered infinite and the bound key is changed accordingly. If a bound value is numerically larger than *dparam.data_tol_bound_wrn*, a warning will be displayed, but the bound is inputted as specified.

Parameters

- `acemode` (*mosek.acemode*) – Defines whether the bound for a constraint (*acemode.con*) or variable (*acemode.var*) is changed. (input)
- `i` (`int`) – Index of the constraint or variable. (input)
- `bk` (*mosek.boundkey*) – New bound key. (input)
- `bl` (`double`) – New lower bound. (input)
- `bu` (`double`) – New upper bound. (input)

Groups *Bound data*

`Task.putboundlist`

```
void putboundlist
(mosek.acemode accmode,
 int[] sub,
 mosek.boundkey[] bk,
 double[] bl,
 double[] bu)
```

Changes the bounds for either some constraints or some variables. If multiple bound changes are specified for a constraint or a variable, only the last change takes effect. Data checks are performed as described in *Task.putbound*.

Parameters

- `acemode` (*mosek.acemode*) – Defines whether bounds for constraints (*acemode.con*) or variables (*acemode.var*) are changed. (input)
- `sub` (`int[]`) – Subscripts of the constraints or variables that should be changed. (input)
- `bk` (*mosek.boundkey*[]) – Bound keys. (input)
- `bl` (`double[]`) – Values for lower bounds. (input)
- `bu` (`double[]`) – Values for upper bounds. (input)

Groups *Bound data*

`Task.putboundslice`

```
void putboundslice
(mosek.accumode con,
 int first,
 int last,
 mosek.boundkey[] bk,
 double[] bl,
 double[] bu)
```

Changes the bounds for a slice of variables or constraints. Data checks are performed as described in *Task.putbound*.

Parameters

- `con` (*mosek.accumode*) – Defines whether bounds for constraints (*accumode.con*) or variables (*accumode.var*) are changed. (input)
- `first` (`int`) – First index in the sequence. (input)
- `last` (`int`) – Last index plus 1 in the sequence. (input)
- `bk` (*mosek.boundkey* []) – Bound keys. (input)
- `bl` (`double` []) – Values for lower bounds. (input)
- `bu` (`double` []) – Values for upper bounds. (input)

Groups *Bound data*

`Task.putcfix`

```
void putcfix (double cfix)
```

Replaces the fixed term in the objective by a new one.

Parameters `cfix` (`double`) – Fixed term in the objective. (input)

Groups *Objective data*

`Task.putcj`

```
void putcj
(int j,
 double cj)
```

Modifies one coefficient in the linear objective vector c , i.e.

$$c_j = cj.$$

If the absolute value exceeds *dparam.data_tol_c_huge* an error is generated. If the absolute value exceeds *dparam.data_tol_cj_large*, a warning is generated, but the coefficient is inputted as specified.

Parameters

- `j` (`int`) – Index of the variable for which c should be changed. (input)
- `cj` (`double`) – New value of c_j . (input)

Groups *Scalar variable data*

`Task.putclist`

```
void putclist
(int[] subj,
 double[] val)
```

Modifies the coefficients in the linear term c in the objective using the principle

$$c_{\text{subj}[t]} = \text{val}[t], \quad t = 0, \dots, \text{num} - 1.$$

If a variable index is specified multiple times in `subj` only the last entry is used. Data checks are performed as in [Task.putcj](#).

Parameters

- `subj` (`int[]`) – Indices of variables for which the coefficient in c should be changed. (input)
- `val` (`double[]`) – New numerical values for coefficients in c that should be modified. (input)

Groups *Scalar variable data*

`Task.putconbound`

```
void putconbound
(int i,
 mosek.boundkey bk,
 double bl,
 double bu)
```

Changes the bounds for one constraint.

If the bound value specified is numerically larger than [dparam.data_tol_bound_inf](#) it is considered infinite and the bound key is changed accordingly. If a bound value is numerically larger than [dparam.data_tol_bound_wrn](#), a warning will be displayed, but the bound is inputted as specified.

Parameters

- `i` (`int`) – Index of the constraint. (input)
- `bk` ([mosek.boundkey](#)) – New bound key. (input)
- `bl` (`double`) – New lower bound. (input)
- `bu` (`double`) – New upper bound. (input)

Groups *Bound data*

`Task.putconboundlist`

```
void putconboundlist
(int[] sub,
 mosek.boundkey[] bk,
 double[] bl,
 double[] bu)
```

Changes the bounds for a list of constraints. If multiple bound changes are specified for a constraint, then only the last change takes effect. Data checks are performed as in [Task.putconbound](#).

Parameters

- `sub` (`int[]`) – List of constraint indexes. (input)
- `bk` ([mosek.boundkey\[\]](#)) – Bound keys. (input)
- `bl` (`double[]`) – Values for lower bounds. (input)
- `bu` (`double[]`) – Values for upper bounds. (input)

Groups *Bound data*

`Task.putconboundslice`

```
void putconboundslice
(int first,
 int last,
 mosek.boundkey[] bk,
 double[] bl,
 double[] bu)
```

Changes the bounds for a slice of the constraints. Data checks are performed as in *Task.putconbound*.

Parameters

- **first** (int) – First index in the sequence. (input)
- **last** (int) – Last index plus 1 in the sequence. (input)
- **bk** (*mosek.boundkey*[]) – Bound keys. (input)
- **bl** (double[]) – Values for lower bounds. (input)
- **bu** (double[]) – Values for upper bounds. (input)

Groups *Linear constraint data, Bound data*

Task.putcone

```
void putcone
(int k,
 mosek.conetype ct,
 double coneapar,
 int[] submem)
```

Replaces a conic constraint.

Parameters

- **k** (int) – Index of the cone. (input)
- **ct** (*mosek.conetype*) – Specifies the type of the cone. (input)
- **coneapar** (double) – This argument is currently not used. It can be set to 0 (input)
- **submem** (int[]) – Variable subscripts of the members in the cone. (input)

Groups *Conic constraint data*

Task.putconename

```
void putconename
(int j,
 String name)
```

Sets the name of a cone.

Parameters

- **j** (int) – Index of the cone. (input)
- **name** (String) – The name of the cone. (input)

Groups *Naming*

Task.putconname

```
void putconname  
(int i,  
 String name)
```

Sets the name of a constraint.

Parameters

- `i` (`int`) – Index of the constraint. (input)
- `name` (`String`) – The name of the constraint. (input)

Groups *Naming*

`Task.putcslice`

```
void putcslice  
(int first,  
 int last,  
 double[] slice)
```

Modifies a slice in the linear term c in the objective using the principle

$$c_j = \text{slice}[j - \text{first}], \quad j = \text{first}, \dots, \text{last} - 1$$

Data checks are performed as in *Task.putcj*.

Parameters

- `first` (`int`) – First element in the slice of c . (input)
- `last` (`int`) – Last element plus 1 of the slice in c to be changed. (input)
- `slice` (`double[]`) – New numerical values for coefficients in c that should be modified. (input)

Groups *Scalar variable data*

`Task.putdoupparam`

```
void putdoupparam  
(mosek.dparam param,  
 double parvalue)
```

Sets the value of a double parameter.

Parameters

- `param` (*`mosek.dparam`*) – Which parameter. (input)
- `parvalue` (`double`) – Parameter value. (input)

Groups *Parameters (put)*

`Task.putintparam`

```
void putintparam  
(mosek.iparam param,  
 int parvalue)
```

Sets the value of an integer parameter.

Please notice that some parameters take values that are defined in Enum classes. This function accepts only integer values, so to use e.g. the value *onoffkey.on*, is necessary to use the member `.value`. For example:

```
task.putintparam(mosek.iparam.opf_write_problem, mosek.onoffkey.on.value)
```

Parameters

- param (*mosek.iparam*) – Which parameter. (input)
- parvalue (int) – Parameter value. (input)

Groups *Parameters (put)*

Task.putmaxnumanz

```
void putmaxnumanz (long maxnumanz)
```

Sets the number of preallocated non-zero entries in A .

MOSEK stores only the non-zero elements in the linear coefficient matrix A and it cannot predict how much storage is required to store A . Using this function it is possible to specify the number of non-zeros to preallocate for storing A .

If the number of non-zeros in the problem is known, it is a good idea to set **maxnumanz** slightly larger than this number, otherwise a rough estimate can be used. In general, if A is inputted in many small chunks, setting this value may speed up the data input phase.

It is not mandatory to call this function, since **MOSEK** will reallocate internal structures whenever it is necessary.

The function call has no effect if both **maxnumcon** and **maxnumvar** are zero.

Parameters **maxnumanz** (long) – Number of preallocated non-zeros in A . (input)

Groups *Scalar variable data*

Task.putmaxnumbarvar

```
void putmaxnumbarvar (int maxnumbarvar)
```

Sets the number of preallocated symmetric matrix variables in the optimization task. When this number of variables is reached **MOSEK** will automatically allocate more space for variables.

It is not mandatory to call this function. It only gives a hint about the amount of data to preallocate for efficiency reasons.

Please note that **maxnumbarvar** must be larger than the current number of symmetric matrix variables in the task.

Parameters **maxnumbarvar** (int) – Number of preallocated symmetric matrix variables. (input)

Groups *Symmetric matrix variable data*

Task.putmaxnumcon

```
void putmaxnumcon (int maxnumcon)
```

Sets the number of preallocated constraints in the optimization task. When this number of constraints is reached **MOSEK** will automatically allocate more space for constraints.

It is never mandatory to call this function, since **MOSEK** will reallocate any internal structures whenever it is required.

Please note that **maxnumcon** must be larger than the current number of constraints in the task.

Parameters `maxnumcon` (`int`) – Number of preallocated constraints in the optimization task. (input)

Groups *Task management*

`Task.putmaxnumcone`

```
void putmaxnumcone (int maxnumcone)
```

Sets the number of preallocated conic constraints in the optimization task. When this number of conic constraints is reached **MOSEK** will automatically allocate more space for conic constraints.

It is not mandatory to call this function, since **MOSEK** will reallocate any internal structures whenever it is required.

Please note that `maxnumcon` must be larger than the current number of conic constraints in the task.

Parameters `maxnumcone` (`int`) – Number of preallocated conic constraints in the optimization task. (input)

Groups *Task management*

`Task.putmaxnumqnz`

```
void putmaxnumqnz (long maxnumqnz)
```

Sets the number of preallocated non-zero entries in quadratic terms.

MOSEK stores only the non-zero elements in Q . Therefore, **MOSEK** cannot predict how much storage is required to store Q . Using this function it is possible to specify the number non-zeros to preallocate for storing Q (both objective and constraints).

It may be advantageous to reserve more non-zeros for Q than actually needed since it may improve the internal efficiency of **MOSEK**, however, it is never worthwhile to specify more than the double of the anticipated number of non-zeros in Q .

It is not mandatory to call this function, since **MOSEK** will reallocate internal structures whenever it is necessary.

Parameters `maxnumqnz` (`long`) – Number of non-zero elements preallocated in quadratic coefficient matrices. (input)

Groups *Scalar variable data*

`Task.putmaxnumvar`

```
void putmaxnumvar (int maxnumvar)
```

Sets the number of preallocated variables in the optimization task. When this number of variables is reached **MOSEK** will automatically allocate more space for variables.

It is not mandatory to call this function. It only gives a hint about the amount of data to preallocate for efficiency reasons.

Please note that `maxnumvar` must be larger than the current number of variables in the task.

Parameters `maxnumvar` (`int`) – Number of preallocated variables in the optimization task. (input)

Groups *Scalar variable data*

`Task.putnadoupparam`


```
void putnadouparam
(String paramname,
 double parvalue)
```

Sets the value of a named double parameter.

Parameters

- `paramname (String)` – Name of a parameter. (input)
- `parvalue (double)` – Parameter value. (input)

Groups *Parameters (put)*

Task.putnaintparam

```
void putnaintparam
(String paramname,
 int parvalue)
```

Sets the value of a named integer parameter.

Parameters

- `paramname (String)` – Name of a parameter. (input)
- `parvalue (int)` – Parameter value. (input)

Groups *Parameters (put)*

Task.putnastrparam

```
void putnastrparam
(String paramname,
 String parvalue)
```

Sets the value of a named string parameter.

Parameters

- `paramname (String)` – Name of a parameter. (input)
- `parvalue (String)` – Parameter value. (input)

Groups *Parameters (put)*

Task.putobjname

```
void putobjname (String objname)
```

Assigns a new name to the objective.

Parameters `objname (String)` – Name of the objective. (input)

Groups *Naming*

Task.putobjsense

```
void putobjsense (mosek.objsense sense)
```

Sets the objective sense of the task.

Parameters `sense (mosek.objsense)` – The objective sense of the task. The values *objsense.maximize* and *objsense.minimize* mean that the problem is maximized or minimized respectively. (input)

Groups *Objective data*`Task.putparam`

```
void putparam
(String parname,
 String parvalue)
```

Checks if `parname` is valid parameter name. If it is, the parameter is assigned the value specified by `parvalue`.

Parameters

- `parname` (`String`) – Parameter name. (input)
- `parvalue` (`String`) – Parameter value. (input)

Groups *Parameters (put)*`Task.putqcon`

```
void putqcon
(int[] qcsbk,
 int[] qcsubi,
 int[] qcsubj,
 double[] qcval)
```

Replace all quadratic entries in the constraints. The list of constraints has the form

$$l_k^c \leq \frac{1}{2} \sum_{i=0}^{numvar-1} \sum_{j=0}^{numvar-1} q_{ij}^k x_i x_j + \sum_{j=0}^{numvar-1} a_{kj} x_j \leq u_k^c, \quad k = 0, \dots, m-1.$$

This function sets all the quadratic terms to zero and then performs the update:

$$q_{qcsubi[t], qcsubj[t]}^{qcsbk[t]} = q_{qcsubj[t], qcsubi[t]}^{qcsbk[t]} = q_{qcsubj[t], qcsubi[t]}^{qcsbk[t]} + qcval[t],$$

for $t = 0, \dots, numqcnz - 1$.

Please note that:

- For large problems it is essential for the efficiency that the function `Task.putmaxnumqcnz` is employed to pre-allocate space.
- Only the lower triangular parts should be specified because the Q matrices are symmetric. Specifying entries where $i < j$ will result in an error.
- Only non-zero elements should be specified.
- The order in which the non-zero elements are specified is insignificant.
- Duplicate elements are added together as shown above. Hence, it is usually not recommended to specify the same entry multiple times.

For a code example see Section 5.5

Parameters

- `qcsbk` (`int[]`) – Constraint subscripts for quadratic coefficients. (input)
- `qcsubi` (`int[]`) – Row subscripts for quadratic constraint matrix. (input)
- `qcsubj` (`int[]`) – Column subscripts for quadratic constraint matrix. (input)
- `qcval` (`double[]`) – Quadratic constraint coefficient values. (input)

Groups *Scalar variable data*

Task.putqconk

```
void putqconk
(int k,
 int[] qcsubi,
 int[] qcsubj,
 double[] qcval)
```

Replaces all the quadratic entries in one constraint. This function performs the same operations as *Task.putqcon* but only with respect to constraint number *k* and it does not modify the other constraints. See the description of *Task.putqcon* for definitions and important remarks.

Parameters

- *k* (int) – The constraint in which the new *Q* elements are inserted. (input)
- *qcsubi* (int[]) – Row subscripts for quadratic constraint matrix. (input)
- *qcsubj* (int[]) – Column subscripts for quadratic constraint matrix. (input)
- *qcval* (double[]) – Quadratic constraint coefficient values. (input)

Groups *Scalar variable data*

Task.putqobj

```
void putqobj
(int[] qosubi,
 int[] qosubj,
 double[] qoval)
```

Replace all quadratic terms in the objective. If the objective has the form

$$\frac{1}{2} \sum_{i=0}^{numvar-1} \sum_{j=0}^{numvar-1} q_{ij}^o x_i x_j + \sum_{j=0}^{numvar-1} c_j x_j + c^f$$

then this function sets all the quadratic terms to zero and then performs the update:

$$q_{qosubi[t], qosubj[t]}^o = q_{qosubj[t], qosubi[t]}^o = q_{qosubj[t], qosubi[t]}^o + qoval[t],$$

for $t = 0, \dots, numqonz - 1$.

See the description of *Task.putqcon* for important remarks and example.

Parameters

- *qosubi* (int[]) – Row subscripts for quadratic objective coefficients. (input)
- *qosubj* (int[]) – Column subscripts for quadratic objective coefficients. (input)
- *qoval* (double[]) – Quadratic objective coefficient values. (input)

Groups *Scalar variable data*

Task.putqobjij

```
void putqobjij
(int i,
 int j,
 double qoij)
```

Replaces one coefficient in the quadratic term in the objective. The function performs the assignment

$$q_{ij}^o = q_{ji}^o = qoij.$$

Only the elements in the lower triangular part are accepted. Setting q_{ij} with $j > i$ will cause an error.

Please note that replacing all quadratic elements one by one is more computationally expensive than replacing them all at once. Use `Task.putqobj` instead whenever possible.

Parameters

- `i` (`int`) – Row index for the coefficient to be replaced. (input)
- `j` (`int`) – Column index for the coefficient to be replaced. (input)
- `qoij` (`double`) – The new value for q_{ij}^o . (input)

Groups *Scalar variable data*

`Task.putskc`

```
void putskc
(mosek.soltype whichsol,
 mosek.stakey[] skc)
```

Sets the status keys for the constraints.

Parameters

- `whichsol` (`mosek.soltype`) – Selects a solution. (input)
- `skc` (`mosek.stakey []`) – Status keys for the constraints. (input)

Groups *Solution (put)*

`Task.putskcslice`

```
void putskcslice
(mosek.soltype whichsol,
 int first,
 int last,
 mosek.stakey[] skc)
```

Sets the status keys for a slice of the constraints.

Parameters

- `whichsol` (`mosek.soltype`) – Selects a solution. (input)
- `first` (`int`) – First index in the sequence. (input)
- `last` (`int`) – Last index plus 1 in the sequence. (input)
- `skc` (`mosek.stakey []`) – Status keys for the constraints. (input)

Groups *Solution (put)*

`Task.putskx`

```
void putskx
(mosek.soltype whichsol,
 mosek.stakey[] skx)
```

Sets the status keys for the scalar variables.

Parameters

- `whichsol` (`mosek.soltype`) – Selects a solution. (input)
- `skx` (`mosek.stakey []`) – Status keys for the variables. (input)

Groups *Solution (put)*

Task.putskxslice

```
void putskxslice
(mosek.soltype whichsol,
 int first,
 int last,
 mosek.stakey[] skx)
```

Sets the status keys for a slice of the variables.

Parameters

- **whichsol** (*mosek.soltype*) – Selects a solution. (input)
- **first** (int) – First index in the sequence. (input)
- **last** (int) – Last index plus 1 in the sequence. (input)
- **skx** (*mosek.stakey*[]) – Status keys for the variables. (input)

Groups *Solution (put)*

Task.putslc

```
void putslc
(mosek.soltype whichsol,
 double[] slc)
```

Sets the s_l^c vector for a solution.

Parameters

- **whichsol** (*mosek.soltype*) – Selects a solution. (input)
- **slc** (double[]) – Dual variables corresponding to the lower bounds on the constraints. (input)

Groups *Solution (put)*

Task.putslcslice

```
void putslcslice
(mosek.soltype whichsol,
 int first,
 int last,
 double[] slc)
```

Sets a slice of the s_l^c vector for a solution.

Parameters

- **whichsol** (*mosek.soltype*) – Selects a solution. (input)
- **first** (int) – First index in the sequence. (input)
- **last** (int) – Last index plus 1 in the sequence. (input)
- **slc** (double[]) – Dual variables corresponding to the lower bounds on the constraints. (input)

Groups *Solution (put)*

Task.putslx

```
void putslx
(mosek.soltype whichsol,
 double[] slx)
```

Sets the s_l^x vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `slx` (`double[]`) – Dual variables corresponding to the lower bounds on the variables. (input)

Groups *Solution (put)*

`Task.putslxslice`

```
void putslxslice
(mosek.soltype whichsol,
 int first,
 int last,
 double[] slx)
```

Sets a slice of the s_l^x vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (`int`) – First index in the sequence. (input)
- `last` (`int`) – Last index plus 1 in the sequence. (input)
- `slx` (`double[]`) – Dual variables corresponding to the lower bounds on the variables. (input)

Groups *Solution (put)*

`Task.putsnx`

```
void putsnx
(mosek.soltype whichsol,
 double[] sux)
```

Sets the s_n^x vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `sux` (`double[]`) – Dual variables corresponding to the upper bounds on the variables. (input)

Groups *Solution (put)*

`Task.putsnxslice`

```
void putsnxslice
(mosek.soltype whichsol,
 int first,
 int last,
 double[] snx)
```

Sets a slice of the s_n^x vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (int) – First index in the sequence. (input)
- `last` (int) – Last index plus 1 in the sequence. (input)
- `snx` (double[]) – Dual variables corresponding to the conic constraints on the variables. (input)

Groups *Solution (put)*`Task.putsolution`

```
void putsolution
(mosek.soltype whichsol,
 mosek.stakey[] skc,
 mosek.stakey[] skx,
 mosek.stakey[] skn,
 double[] xc,
 double[] xx,
 double[] y,
 double[] slc,
 double[] suc,
 double[] slx,
 double[] sux,
 double[] snx)
```

Inserts a solution into the task.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `skc` (*mosek.stakey*[]) – Status keys for the constraints. (input)
- `skx` (*mosek.stakey*[]) – Status keys for the variables. (input)
- `skn` (*mosek.stakey*[]) – Status keys for the conic constraints. (input)
- `xc` (double[]) – Primal constraint solution. (input)
- `xx` (double[]) – Primal variable solution. (input)
- `y` (double[]) – Vector of dual variables corresponding to the constraints. (input)
- `slc` (double[]) – Dual variables corresponding to the lower bounds on the constraints. (input)
- `suc` (double[]) – Dual variables corresponding to the upper bounds on the constraints. (input)
- `slx` (double[]) – Dual variables corresponding to the lower bounds on the variables. (input)
- `sux` (double[]) – Dual variables corresponding to the upper bounds on the variables. (input)
- `snx` (double[]) – Dual variables corresponding to the conic constraints on the variables. (input)

Groups *Solution (put)*`Task.putsolutioni`

```
void putsolutioni
(mosek.accmode accmode,
 int i,
 mosek.soltype whichsol,
 mosek.stakey sk,
 double x,
 double sl,
 double su,
 double sn)
```

Sets the primal and dual solution information for a single constraint or variable.

Parameters

- `accmode` (*mosek.accmode*) – Defines whether solution information for a constraint (*accmode.con*) or for a variable (*accmode.var*) is modified. (input)
- `i` (`int`) – Index of the constraint or variable. (input)
- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `sk` (*mosek.stakey*) – Status key of the constraint or variable. (input)
- `x` (`double`) – Solution value of the primal constraint or variable. (input)
- `sl` (`double`) – Solution value of the dual variable associated with the lower bound. (input)
- `su` (`double`) – Solution value of the dual variable associated with the upper bound. (input)
- `sn` (`double`) – Solution value of the dual variable associated with the conic constraint. (input)

Groups *Solution* (*put*)

Task.putsolutionyi

```
void putsolutionyi
(int i,
 mosek.soltype whichsol,
 double y)
```

Inputs the dual variable of a solution.

Parameters

- `i` (`int`) – Index of the dual variable. (input)
- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `y` (`double`) – Solution value of the dual variable. (input)

Task.putstrparam

```
void putstrparam
(mosek.sparam param,
 String parvalue)
```

Sets the value of a string parameter.

Parameters

- `param` (*mosek.sparam*) – Which parameter. (input)
- `parvalue` (`String`) – Parameter value. (input)

Groups *Parameters (put)*

Task.putsuc

```
void putsuc
(mosek.soltype whichsol,
 double[] suc)
```

Sets the s_u^c vector for a solution.

Parameters

- **whichsol** (*mosek.soltype*) – Selects a solution. (input)
- **suc** (double[]) – Dual variables corresponding to the upper bounds on the constraints. (input)

Groups *Solution (put)*

Task.putsucslice

```
void putsucslice
(mosek.soltype whichsol,
 int first,
 int last,
 double[] suc)
```

Sets a slice of the s_u^c vector for a solution.

Parameters

- **whichsol** (*mosek.soltype*) – Selects a solution. (input)
- **first** (int) – First index in the sequence. (input)
- **last** (int) – Last index plus 1 in the sequence. (input)
- **suc** (double[]) – Dual variables corresponding to the upper bounds on the constraints. (input)

Groups *Solution (put)*

Task.putsuc

```
void putsuc
(mosek.soltype whichsol,
 double[] suc)
```

Sets the s_u^x vector for a solution.

Parameters

- **whichsol** (*mosek.soltype*) – Selects a solution. (input)
- **suc** (double[]) – Dual variables corresponding to the upper bounds on the variables. (input)

Groups *Solution (put)*

Task.putsucslice

```
void putsucslice
(mosek.soltype whichsol,
 int first,
```

```
int last,  
double[] sux)
```

Sets a slice of the s_u^x vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (int) – First index in the sequence. (input)
- `last` (int) – Last index plus 1 in the sequence. (input)
- `sux` (double[]) – Dual variables corresponding to the upper bounds on the variables. (input)

Groups *Solution* (*put*)

`Task.puttaskname`

```
void puttaskname (String taskname)
```

Assigns a new name to the task.

Parameters `taskname` (String) – Name assigned to the task. (input)

Groups *Naming*

`Task.putvarbound`

```
void putvarbound  
(int j,  
 mosek.boundkey bk,  
 double bl,  
 double bu)
```

Changes the bounds for one variable.

If the bound value specified is numerically larger than *dparam.data_tol_bound_inf* it is considered infinite and the bound key is changed accordingly. If a bound value is numerically larger than *dparam.data_tol_bound_wrn*, a warning will be displayed, but the bound is inputted as specified.

Parameters

- `j` (int) – Index of the variable. (input)
- `bk` (*mosek.boundkey*) – New bound key. (input)
- `bl` (double) – New lower bound. (input)
- `bu` (double) – New upper bound. (input)

Groups *Bound data*

`Task.putvarboundlist`

```
void putvarboundlist  
(int[] sub,  
 mosek.boundkey[] bks,  
 double[] bls,  
 double[] bus)
```

Changes the bounds for one or more variables. If multiple bound changes are specified for a variable, then only the last change takes effect. Data checks are performed as in *Task.putvarbound*.

Parameters

- `sub (int[])` – List of variable indexes. (input)
- `bkx (mosek.boundkey [])` – Bound keys for the variables. (input)
- `blx (double[])` – Lower bounds for the variables. (input)
- `bux (double[])` – Upper bounds for the variables. (input)

Groups *Bound data*

`Task.putvarboundslice`

```
void putvarboundslice
(int first,
 int last,
 mosek.boundkey[] bk,
 double[] bl,
 double[] bu)
```

Changes the bounds for a slice of the variables. Data checks are performed as in *Task.putvarbound*.

Parameters

- `first (int)` – First index in the sequence. (input)
- `last (int)` – Last index plus 1 in the sequence. (input)
- `bk (mosek.boundkey [])` – Bound keys. (input)
- `bl (double[])` – Values for lower bounds. (input)
- `bu (double[])` – Values for upper bounds. (input)

Groups *Scalar variable data*

`Task.putvarname`

```
void putvarname
(int j,
 String name)
```

Sets the name of a variable.

Parameters

- `j (int)` – Index of the variable. (input)
- `name (String)` – The variable name. (input)

Groups *Naming*

`Task.putvartype`

```
void putvartype
(int j,
 mosek.variabletype vartype)
```

Sets the variable type of one variable.

Parameters

- `j (int)` – Index of the variable. (input)
- `vartype (mosek.variabletype)` – The new variable type. (input)

Groups *Scalar variable data*

Task.putvartypelist

```
void putvartypelist
(int[] subj,
 mosek.variabletype[] vartype)
```

Sets the variable type for one or more variables. If the same index is specified multiple times in `subj` only the last entry takes effect.

Parameters

- `subj` (`int[]`) – A list of variable indexes for which the variable type should be changed. (input)
- `vartype` (`mosek.variabletype[]`) – A list of variable types that should be assigned to the variables specified by `subj`. (input)

Groups *Scalar variable data*

Task.putxc

```
void putxc
(mosek.soltype whichsol,
 double[] xc)
```

Sets the x^c vector for a solution.

Parameters

- `whichsol` (`mosek.soltype`) – Selects a solution. (input)
- `xc` (`double[]`) – Primal constraint solution. (output)

Groups *Solution (put)*

Task.putxcslice

```
void putxcslice
(mosek.soltype whichsol,
 int first,
 int last,
 double[] xc)
```

Sets a slice of the x^c vector for a solution.

Parameters

- `whichsol` (`mosek.soltype`) – Selects a solution. (input)
- `first` (`int`) – First index in the sequence. (input)
- `last` (`int`) – Last index plus 1 in the sequence. (input)
- `xc` (`double[]`) – Primal constraint solution. (input)

Groups *Solution (put)*

Task.putxx

```
void putxx
(mosek.soltype whichsol,
 double[] xx)
```

Sets the x^x vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `xx` (`double[]`) – Primal variable solution. (input)

Groups *Solution (put)*`Task.putxxslice`

```
void putxxslice
(mosek.soltype whichsol,
 int first,
 int last,
 double[] xx)
```

Obtains a slice of the x^x vector for a solution.**Parameters**

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (`int`) – First index in the sequence. (input)
- `last` (`int`) – Last index plus 1 in the sequence. (input)
- `xx` (`double[]`) – Primal variable solution. (input)

Groups *Solution (put)*`Task.puty`

```
void puty
(mosek.soltype whichsol,
 double[] y)
```

Sets the y vector for a solution.**Parameters**

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `y` (`double[]`) – Vector of dual variables corresponding to the constraints. (input)

Groups *Solution (put)*`Task.putyslice`

```
void putyslice
(mosek.soltype whichsol,
 int first,
 int last,
 double[] y)
```

Sets a slice of the y vector for a solution.**Parameters**

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (`int`) – First index in the sequence. (input)
- `last` (`int`) – Last index plus 1 in the sequence. (input)
- `y` (`double[]`) – Vector of dual variables corresponding to the constraints. (input)

Groups *Solution (put)*

Task.readdata

```
void readdata (String filename)
```

Reads an optimization problem and associated data from a file.

Parameters filename (String) – A valid file name. (input)

Groups *Data file*

Task.readdataformat

```
void readdataformat  
(String filename,  
 mosek.dataformat format,  
 mosek.compresstype compress)
```

Reads an optimization problem and associated data from a file.

Parameters

- filename (String) – A valid file name. (input)
- format (*mosek.dataformat*) – File data format. (input)
- compress (*mosek.compresstype*) – File compression type. (input)

Groups *Data file*

Task.readparamfile

```
void readparamfile (String filename)
```

Reads **MOSEK** parameters from a file. Data is read from the file **filename** if it is a nonempty string. Otherwise data is read from the file specified by *sparam.param_read_file_name*.

Parameters filename (String) – A valid file name. (input)

Groups *Data file*

Task.readsolution

```
void readsolution  
(mosek.soltype whichsol,  
 String filename)
```

Reads a solution file and inserts it as a specified solution in the task. Data is read from the file **filename** if it is a nonempty string. Otherwise data is read from one of the files specified by *sparam.bas_sol_file_name*, *sparam.itr_sol_file_name* or *sparam.int_sol_file_name* depending on which solution is chosen.

Parameters

- whichsol (*mosek.soltype*) – Selects a solution. (input)
- filename (String) – A valid file name. (input)

Groups *Data file*

Task.readsummary

```
void readsummary (mosek.streamtype whichstream)
```

Prints a short summary of last file that was read.

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

Groups *Task diagnostics*

`Task.readtask`

```
void readtask (String filename)
```

Load task data from a file, replacing any data that already exists in the task object. All problem data, parameters and other settings are resorted, but if the file contains solutions, the solution status after loading a file is set to unknown, even if it was optimal or otherwise well-defined when the file was dumped.

See section 19.6 for a description of the Task format.

Parameters `filename` (String) – A valid file name. (input)

`Task.removebarvars`

```
void removebarvars (int[] subset)
```

The function removes a subset of the symmetric matrices from the optimization task. This implies that the remaining symmetric matrices are renumbered.

Parameters `subset` (int[]) – Indexes of symmetric matrices which should be removed. (input)

Groups *Symmetric matrix variable data*

`Task.removecones`

```
void removecones (int[] subset)
```

Removes a number of conic constraints from the problem. This implies that the remaining conic constraints are renumbered. In general, it is much more efficient to remove a cone with a high index than a low index.

Parameters `subset` (int[]) – Indexes of cones which should be removed. (input)

Groups *Conic constraint data*

`Task.removecons`

```
void removecons (int[] subset)
```

The function removes a subset of the constraints from the optimization task. This implies that the remaining constraints are renumbered.

Parameters `subset` (int[]) – Indexes of constraints which should be removed. (input)

Groups *Linear constraint data*

`Task.removevars`

```
void removevars (int[] subset)
```

The function removes a subset of the variables from the optimization task. This implies that the remaining variables are renumbered.

Parameters `subset` (int[]) – Indexes of variables which should be removed. (input)

Groups *Scalar variable data***Task.resizetask**

```
void resizetask
(int maxnumcon,
 int maxnumvar,
 int maxnumcone,
 long maxnumanz,
 long maxnumqnz)
```

Sets the amount of preallocated space assigned for each type of data in an optimization task.

It is never mandatory to call this function, since it only gives a hint about the amount of data to preallocate for efficiency reasons.

Please note that the procedure is **destructive** in the sense that all existing data stored in the task is destroyed.

Parameters

- **maxnumcon** (int) – New maximum number of constraints. (input)
- **maxnumvar** (int) – New maximum number of variables. (input)
- **maxnumcone** (int) – New maximum number of cones. (input)
- **maxnumanz** (long) – New maximum number of non-zeros in A . (input)
- **maxnumqnz** (long) – New maximum number of non-zeros in all Q matrices. (input)

Task.sensitivityreport

```
void sensitivityreport (mosek.streamtype whichstream)
```

Reads a sensitivity format file from a location given by *sparam.sensitivity_file_name* and writes the result to the stream **whichstream**. If *sparam.sensitivity_res_file_name* is set to a non-empty string, then the sensitivity report is also written to a file of this name.

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

Groups *Sensitivity analysis***Task.set_InfoCallback**

```
void set_InfoCallback (mosek.DataCallback callback)
```

Receive callbacks with solver status and information during optimization.

For example:

```
task.set_InfoCallback(
    new mosek.InfoCallback() {
        int callback(mosek.callbackcode code, double[] dinf, int[] iinf, long[] liinf) {
            System.println("Callback "+code+", intpnt time : "+dinf[mosek.dinfitem.intpnt_time.
↪getValue()]);
            return 0;
        } } );
```

Parameters **callback** (*DataCallback*) – The callback object. (input)

Task.set_ItgSolutionCallback


```
void set_ItgSolutionCallback (mosek.ItgSolutionCallback callback)
```

Receive callbacks with solution updates from the mixed-integer optimizer.

For example:

```
task.set_ItgSolutionCallback(
    new mosek.ItgSolutionCallback() {
        void callback(double[] xx) {
            System.out.print("New integer solution: ");
            for (double v : xx) System.out.print(" " + v + " ");
            System.out.println("");
        }
    });
```

Parameters `callback` (*ItgSolutionCallback*) – The callback object. (input)

`Task.set_Progress`

```
void set_Progress (mosek.Progress callback)
```

Receive callbacks about current status of the solver during optimization.

For example:

```
task.set_Progress(new mosek.Progress() { int progress(mosek.callbackcode code) { System.
    println("Callback "+code); return 0; } });
```

Parameters `callback` (*Progress*) – The callback object. (input)

`Task.set_Stream`

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

Directs all output from a task stream to a callback object.

Can for example be called as:

```
task.set_Stream(mosek.streamtype.log, new Stream() { public void stream(String s) {
    System.out.print(s); } });
```

Parameters

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

`Task.setdefault`

```
void setdefaults ()
```

Resets all the parameters to their default values.

Groups *Parameter management*

`Task.solutiondef`

```
void solutiondef
(mosek.soltype whichsol,
 boolean[] isdef)
```

```
boolean solutiondef (mosek.soltype whichsol)
```

Checks whether a solution is defined.

Parameters

- **whichsol** (*mosek.soltype*) – Selects a solution. (input)
- **isdef** (boolean *by reference*) – Is non-zero if the requested solution is defined. (output)

Return (boolean) – Is non-zero if the requested solution is defined.

Groups *Solution information*

Task.solutionsummary

```
void solutionsummary (mosek.streamtype whichstream)
```

Prints a short summary of the current solutions.

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

Groups *Task diagnostics*

Task.solvewithbasis

```
void solvewithbasis
(int transp,
 int[] numnz,
 int[] sub,
 double[] val)
```

```
int solvewithbasis
(int transp,
 int numnz,
 int[] sub,
 double[] val)
```

If a basic solution is available, then exactly *numcon* basis variables are defined. These *numcon* basis variables are denoted the basis. Associated with the basis is a basis matrix denoted B . This function solves either the linear equation system

$$B\overline{X} = b \quad (18.3)$$

or the system

$$B^T\overline{X} = b \quad (18.4)$$

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.initbasis**solve* an ordering of the basis variables is obtained, which can be used to deduce how **MOSEK** has constructed B . Indeed if the k -th basis variable is variable x_j it implies that

$$B_{i,k} = A_{i,j}, \quad i = 0, \dots, \text{numcon} - 1.$$

Otherwise if the k -th basis variable is variable x_j^c it implies that

$$B_{i,k} = \begin{cases} -1, & i = j, \\ 0, & i \neq j. \end{cases}$$

The function `Task.initbasissolve` must be called before a call to this function. Please note that this function exploits the sparsity in the vector b to speed up the computations.

Parameters

- `transp (int)` – If this argument is zero, then (18.3) is solved, if non-zero then (18.4) is solved. (input)
- `numnz (int by reference)` – As input it is the number of non-zeros in b . As output it is the number of non-zeros in \bar{X} . (input/output)
- `numnz (int)` – As input it is the number of non-zeros in b . As output it is the number of non-zeros in \bar{X} . (input/output)
- `sub (int[])` – As input it contains the positions of non-zeros in b . As output it contains the positions of the non-zeros in \bar{X} . It must have room for *numcon* elements. (input/output)
- `val (double[])` – As input it is the vector b as a dense vector (although the positions of non-zeros are specified in `sub` it is required that `val[i] = 0` when $b[i] = 0$). As output `val` is the vector \bar{X} as a dense vector. It must have length *numcon*. (input/output)

Return (int) – As input it is the number of non-zeros in b . As output it is the number of non-zeros in \bar{X} .

Groups *Basis matrix*

Task.strtoconetype

```
void strtoconetype
(String str,
 mosek.conetype[] conetype)
```

Obtains cone type code corresponding to a cone type string.

Parameters

- `str (String)` – String corresponding to the cone type code `conetype`. (input)
- `conetype (mosek.conetype by reference)` – The cone type corresponding to the string `str`. (output)

Task.strtosk

```
void strtosk
(String str,
 int[] sk)
```

Obtains the status key corresponding to an explanatory string.

Parameters

- `str (String)` – Status key string. (input)
- `sk (int by reference)` – Status key corresponding to the string. (output)

Task.toconic

```
void toconic ()
```

This function tries to reformulate a given Quadratically Constrained Quadratic Optimization problem (QCQP) as a Conic Quadratic Optimization problem (CQO). The first step of the reformulation is to convert the quadratic term of the objective function, if any, into a constraint. Then the following steps are repeated for each quadratic constraint:

- a conic constraint is added along with a suitable number of auxiliary variables and constraints;
- the original quadratic constraint is not removed, but all its coefficients are zeroed out.

Note that the reformulation preserves all the original variables.

The conversion is performed in-place, i.e. the task passed as argument is modified on exit. That also means that if the reformulation fails, i.e. the given QCQP is not representable as a CQO, then the task has an undefined state. In some cases, users may want to clone the task to ensure a clean copy is preserved.

Task.unset_Progress

```
void unset_Progress ()
```

Deactivates all user callback functions.

Task.updatesolutioninfo

```
void updatesolutioninfo (mosek.soltype whichsol)
```

Update the information items related to the solution.

Parameters *whichsol* (*mosek.soltype*) – Selects a solution. (input)

Groups *Task diagnostics*

Task.writedata

```
void writedata (String filename)
```

Writes problem data associated with the optimization task to a file in one of the supported formats. See Section 19 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. To write in compressed format append the extension *.gz*. E.g to write a gzip compressed MPS file use the extension *mps.gz*.

Please note that MPS, LP and OPF files require all variables to have unique names. If a task contains no names, it is possible to write the file with automatically generated anonymous names by setting the *iparam.write_generic_names* parameter to *onoffkey.on*.

Data is written to the file *filename* if it is a nonempty string. Otherwise data is written to the file specified by *sparam.data_file_name*.

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 *filename* (String) – A valid file name. (input)

Groups *Data file*

Task.writejsonsol

```
void writejsonsol (String filename)
```

Saves the current solutions and solver information items in a JSON file.

Parameters filename (String) – A valid file name. (input)

Groups *Data file*

Task.writeparamfile

```
void writeparamfile (String filename)
```

Writes all the parameters to a parameter file.

Parameters filename (String) – A valid file name. (input)

Groups *Data file*

Task.writesolution

```
void writesolution
(mosek.soltype whichsol,
String filename)
```

Saves the current basic, interior-point, or integer solution to a file.

Parameters

- whichsol (*mosek.soltype*) – Selects a solution. (input)
- filename (String) – A valid file name. (input)

Groups *Data file*

Task.writetask

```
void writetask (String filename)
```

Write a binary dump of the task data. This format saves all problem data, coefficients and parameter settings but does not save callback functions and general non-linear terms.

See section 19.6 for a description of the Task format.

Parameters filename (String) – A valid file name. (input)

18.5 Exceptions

MosekException

The base class for all exceptions in MOSEK.

Exception

Base class for exceptions that correspond to MOSEK response codes.

Implements *MosekException*

Error

Exception class used for all error response codes from MOSEK.

Implements *Exception*

Warning

Exception class used for all warning response codes from MOSEK.

Implements *Exception*

NullArrayException

Exception thrown when null was passed to a method that expected non-null array argument.

Implements *MosekException*

ArrayLengthException

Exception thrown the length of an array was smaller than required. This will happen, for example, if requesting a list of N values, but the array passed to the method is less than N elements long.

Implements *MosekException*

18.6 Parameters grouped by topic

Analysis

- *dparam.ana_sol_infeas_tol*
- *iparam.ana_sol_basis*
- *iparam.ana_sol_print_violated*
- *iparam.log_ana_pro*

Basis identification

- *dparam.sim_lu_tol_rel_piv*
- *iparam.bi_clean_optimizer*
- *iparam.bi_ignore_max_iter*
- *iparam.bi_ignore_num_error*
- *iparam.bi_max_iterations*
- *iparam.intpnt_basis*
- *iparam.log_bi*
- *iparam.log_bi_freq*

Conic interior-point method

- *dparam.intpnt_co_tol_dfeas*
- *dparam.intpnt_co_tol_infeas*
- *dparam.intpnt_co_tol_mu_red*
- *dparam.intpnt_co_tol_near_rel*
- *dparam.intpnt_co_tol_pfeas*
- *dparam.intpnt_co_tol_rel_gap*

Data check

- *dparam.data_sym_mat_tol*
- *dparam.data_sym_mat_tol_huge*
- *dparam.data_sym_mat_tol_large*
- *dparam.data_tol_aij*
- *dparam.data_tol_aij_huge*

- *dparam.data_tol_aij_large*
- *dparam.data_tol_bound_inf*
- *dparam.data_tol_bound_wrn*
- *dparam.data_tol_c_huge*
- *dparam.data_tol_cj_large*
- *dparam.data_tol_qij*
- *dparam.data_tol_x*
- *dparam.semidefinite_tol_approx*
- *iparam.check_convexity*
- *iparam.log_check_convexity*

Data input/output

- *iparam.infeas_report_auto*
- *iparam.log_file*
- *iparam.opf_max_terms_per_line*
- *iparam.opf_write_header*
- *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*
- *iparam.param_read_case_name*
- *iparam.param_read_ign_error*
- *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*
- *iparam.read_mps_format*
- *iparam.read_mps_width*
- *iparam.read_task_ignore_param*
- *iparam.sol_read_name_width*
- *iparam.sol_read_width*
- *iparam.write_bas_constraints*
- *iparam.write_bas_head*
- *iparam.write_bas_variables*

- *iparam.write_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*
- *iparam.write_lp_line_width*
- *iparam.write_lp_quoted_names*
- *iparam.write_lp_strict_format*
- *iparam.write_lp_terms_per_line*
- *iparam.write_mps_format*
- *iparam.write_mps_int*
- *iparam.write_precision*
- *iparam.write_sol_barvariables*
- *iparam.write_sol_constraints*
- *iparam.write_sol_head*
- *iparam.write_sol_ignore_invalid_names*
- *iparam.write_sol_variables*
- *iparam.write_task_inc_sol*
- *iparam.write_xml_mode*
- *sparam.bas_sol_file_name*
- *sparam.data_file_name*
- *sparam.debug_file_name*
- *sparam.int_sol_file_name*
- *sparam.itr_sol_file_name*
- *sparam.mio_debug_string*
- *sparam.param_comment_sign*
- *sparam.param_read_file_name*
- *sparam.param_write_file_name*
- *sparam.read_mps_bou_name*
- *sparam.read_mps_obj_name*
- *sparam.read_mps_ran_name*
- *sparam.read_mps_rhs_name*
- *sparam.sensitivity_file_name*

- *sparam.sensitivity_res_file_name*
- *sparam.sol_filter_xc_low*
- *sparam.sol_filter_xc_upr*
- *sparam.sol_filter_xx_low*
- *sparam.sol_filter_xx_upr*
- *sparam.stat_file_name*
- *sparam.stat_key*
- *sparam.stat_name*
- *sparam.write_lp_gen_var_name*

Debugging

- *iparam.auto_sort_a_before_opt*

Dual simplex

- *iparam.sim_dual_crash*
- *iparam.sim_dual_restrict_selection*
- *iparam.sim_dual_selection*

Infeasibility report

- *iparam.infeas_generic_names*
- *iparam.infeas_report_level*
- *iparam.log_infeas_ana*

Interior-point method

- *dparam.check_convexity_rel_tol*
- *dparam.intpnt_co_tol_dfeas*
- *dparam.intpnt_co_tol_infeas*
- *dparam.intpnt_co_tol_mu_red*
- *dparam.intpnt_co_tol_near_rel*
- *dparam.intpnt_co_tol_pfeas*
- *dparam.intpnt_co_tol_rel_gap*
- *dparam.intpnt_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_qo_tol_dfeas*
- *dparam.intpnt_qo_tol_infeas*
- *dparam.intpnt_qo_tol_mu_red*
- *dparam.intpnt_qo_tol_near_rel*
- *dparam.intpnt_qo_tol_pfeas*
- *dparam.intpnt_qo_tol_rel_gap*
- *dparam.intpnt_tol_dfeas*
- *dparam.intpnt_tol_dsafe*
- *dparam.intpnt_tol_infeas*
- *dparam.intpnt_tol_mu_red*
- *dparam.intpnt_tol_path*
- *dparam.intpnt_tol_pfeas*
- *dparam.intpnt_tol_psafe*
- *dparam.intpnt_tol_rel_gap*
- *dparam.intpnt_tol_rel_step*
- *dparam.intpnt_tol_step_size*
- *dparam.qcgo_reformulate_rel_drop_tol*
- *iparam.bi_ignore_max_iter*
- *iparam.bi_ignore_num_error*
- *iparam.intpnt_basis*
- *iparam.intpnt_diff_step*
- *iparam.intpnt_hotstart*
- *iparam.intpnt_max_iterations*
- *iparam.intpnt_max_num_cor*
- *iparam.intpnt_max_num_refinement_steps*
- *iparam.intpnt_off_col_trh*
- *iparam.intpnt_order_method*
- *iparam.intpnt_regularization_use*
- *iparam.intpnt_scaling*
- *iparam.intpnt_solve_form*
- *iparam.intpnt_starting_point*
- *iparam.log_intpnt*

License manager

- *iparam.cache_license*
- *iparam.license_debug*
- *iparam.license_pause_time*
- *iparam.license_suppress_expire_wrns*
- *iparam.license_trh_expiry_wrn*

- *iparam.license_wait*

Logging

- *iparam.log*
- *iparam.log_ana_pro*
- *iparam.log_bi*
- *iparam.log_bi_freq*
- *iparam.log_cut_second_opt*
- *iparam.log_expand*
- *iparam.log_feas_repair*
- *iparam.log_file*
- *iparam.log_infeas_ana*
- *iparam.log_intpnt*
- *iparam.log_mio*
- *iparam.log_mio_freq*
- *iparam.log_order*
- *iparam.log_presolve*
- *iparam.log_response*
- *iparam.log_sensitivity*
- *iparam.log_sensitivity_opt*
- *iparam.log_sim*
- *iparam.log_sim_freq*
- *iparam.log_storage*

Mixed-integer optimization

- *dparam.mio_disable_term_time*
- *dparam.mio_max_time*
- *dparam.mio_near_tol_abs_gap*
- *dparam.mio_near_tol_rel_gap*
- *dparam.mio_rel_gap_const*
- *dparam.mio_tol_abs_gap*
- *dparam.mio_tol_abs_relax_int*
- *dparam.mio_tol_feas*
- *dparam.mio_tol_rel_dual_bound_improvement*
- *dparam.mio_tol_rel_gap*
- *iparam.log_mio*
- *iparam.log_mio_freq*
- *iparam.mio_branch_dir*

- *iparam.mio_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*
- *iparam.mio_heuristic_level*
- *iparam.mio_max_num_branches*
- *iparam.mio_max_num_relaxs*
- *iparam.mio_max_num_solutions*
- *iparam.mio_node_optimizer*
- *iparam.mio_node_selection*
- *iparam.mio_perspective_reformulate*
- *iparam.mio_probing_level*
- *iparam.mio_rins_max_nodes*
- *iparam.mio_root_optimizer*
- *iparam.mio_root_repeat_presolve_level*
- *iparam.mio_vb_detection_level*

Nonlinear convex method

- *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.check_convexity*
- *iparam.log_check_convexity*

Output information

- *iparam.infeas_report_level*
- *iparam.license_suppress_expire_wrns*
- *iparam.license_trh_expiry_wrn*
- *iparam.log*
- *iparam.log_bi*

- *iparam.log_bi_freq*
- *iparam.log_cut_second_opt*
- *iparam.log_expand*
- *iparam.log_feas_repair*
- *iparam.log_file*
- *iparam.log_infeas_ana*
- *iparam.log_intpnt*
- *iparam.log_mio*
- *iparam.log_mio_freq*
- *iparam.log_order*
- *iparam.log_response*
- *iparam.log_sensitivity*
- *iparam.log_sensitivity_opt*
- *iparam.log_sim*
- *iparam.log_sim_freq*
- *iparam.log_sim_minor*
- *iparam.log_storage*
- *iparam.max_num_warnings*

Overall solver

- *iparam.bi_clean_optimizer*
- *iparam.infeas_prefer_primal*
- *iparam.license_wait*
- *iparam.mio_mode*
- *iparam.optimizer*
- *iparam.presolve_level*
- *iparam.presolve_max_num_reductions*
- *iparam.presolve_use*
- *iparam.primal_repair_optimizer*
- *iparam.sensitivity_all*
- *iparam.sensitivity_optimizer*
- *iparam.sensitivity_type*
- *iparam.solution_callback*

Overall system

- *iparam.auto_update_sol_info*
- *iparam.intpnt_multi_thread*
- *iparam.license_wait*

- *iparam.log_storage*
- *iparam.mio_mt_user_cb*
- *iparam.mt_spincount*
- *iparam.num_threads*
- *iparam.remove_unused_solutions*
- *iparam.timing_level*
- *sparam.remote_access_token*

Presolve

- *dparam.presolve_tol_abs_lindep*
- *dparam.presolve_tol_aij*
- *dparam.presolve_tol_rel_lindep*
- *dparam.presolve_tol_s*
- *dparam.presolve_tol_x*
- *iparam.presolve_eliminator_max_fill*
- *iparam.presolve_eliminator_max_num_tries*
- *iparam.presolve_level*
- *iparam.presolve_lindep_abs_work_trh*
- *iparam.presolve_lindep_rel_work_trh*
- *iparam.presolve_lindep_use*
- *iparam.presolve_max_num_reductions*
- *iparam.presolve_use*

Primal simplex

- *iparam.sim_primal_crash*
- *iparam.sim_primal_restrict_selection*
- *iparam.sim_primal_selection*

Progress callback

- *iparam.solution_callback*

Simplex optimizer

- *dparam.basis_rel_tol_s*
- *dparam.basis_tol_s*
- *dparam.basis_tol_x*
- *dparam.sim_lu_tol_rel_piv*
- *dparam.simplex_abs_tol_piv*
- *iparam.basis_solve_use_plus_one*

- *iparam.log_sim*
- *iparam.log_sim_freq*
- *iparam.log_sim_minor*
- *iparam.sensitivity_optimizer*
- *iparam.sim_basis_factor_use*
- *iparam.sim_degen*
- *iparam.sim_dual_phaseone_method*
- *iparam.sim_exploit_dupvec*
- *iparam.sim_hotstart*
- *iparam.sim_hotstart_lu*
- *iparam.sim_max_iterations*
- *iparam.sim_max_num_setbacks*
- *iparam.sim_non_singular*
- *iparam.sim_primal_phaseone_method*
- *iparam.sim_refactor_freq*
- *iparam.sim_reformulation*
- *iparam.sim_save_lu*
- *iparam.sim_scaling*
- *iparam.sim_scaling_method*
- *iparam.sim_solve_form*
- *iparam.sim_stability_priority*
- *iparam.sim_switch_optimizer*

Solution input/output

- *iparam.infeas_report_auto*
- *iparam.sol_filter_keep_basic*
- *iparam.sol_filter_keep_ranged*
- *iparam.sol_read_name_width*
- *iparam.sol_read_width*
- *iparam.write_bas_constraints*
- *iparam.write_bas_head*
- *iparam.write_bas_variables*
- *iparam.write_int_constraints*
- *iparam.write_int_head*
- *iparam.write_int_variables*
- *iparam.write_sol_barvariables*
- *iparam.write_sol_constraints*
- *iparam.write_sol_head*
- *iparam.write_sol_ignore_invalid_names*

- *iparam.write_sol_variables*
- *sparam.bas_sol_file_name*
- *sparam.int_sol_file_name*
- *sparam.itr_sol_file_name*
- *sparam.sol_filter_xc_low*
- *sparam.sol_filter_xc_upr*
- *sparam.sol_filter_xx_low*
- *sparam.sol_filter_xx_upr*

Termination criteria

- *dparam.basis_rel_tol_s*
- *dparam.basis_tol_s*
- *dparam.basis_tol_x*
- *dparam.intpnt_co_tol_dfeas*
- *dparam.intpnt_co_tol_infeas*
- *dparam.intpnt_co_tol_mu_red*
- *dparam.intpnt_co_tol_near_rel*
- *dparam.intpnt_co_tol_pfeas*
- *dparam.intpnt_co_tol_rel_gap*
- *dparam.intpnt_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_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*
- *dparam.mio_max_time*

- *dparam.mio_near_tol_rel_gap*
- *dparam.mio_rel_gap_const*
- *dparam.mio_tol_rel_gap*
- *dparam.optimizer_max_time*
- *dparam.upper_obj_cut*
- *dparam.upper_obj_cut_finite_trh*
- *iparam.bi_max_iterations*
- *iparam.intpnt_max_iterations*
- *iparam.mio_max_num_branches*
- *iparam.mio_max_num_solutions*
- *iparam.sim_max_iterations*

Other

- *iparam.compress_statfile*

18.7 Parameters (alphabetical list sorted by type)

- *Double parameters*
- *Integer parameters*
- *String parameters*

18.7.1 Double parameters

dparam

The enumeration type containing all double parameters.

dparam.ana_sol_infeas_tol

If a constraint violates its bound with an amount larger than this value, the constraint name, index and violation will be printed by the solution analyzer.

Default 1e-6

Accepted [0.0; +inf]

Groups *Analysis*

dparam.basis_rel_tol_s

Maximum relative dual bound violation allowed in an optimal basic solution.

Default 1.0e-12

Accepted [0.0; +inf]

Groups *Simplex optimizer, Termination criteria*

dparam.basis_tol_s

Maximum absolute dual bound violation in an optimal basic solution.

Default 1.0e-6

Accepted [1.0e-9; +inf]

Groups *Simplex optimizer, Termination criteria*

`dparam.basis_tol_x`

Maximum absolute primal bound violation allowed in an optimal basic solution.

Default 1.0e-6

Accepted [1.0e-9; +inf]

Groups *Simplex optimizer, Termination criteria*

`dparam.check_convexity_rel_tol`

This parameter controls when the full convexity check declares a problem to be non-convex. Increasing this tolerance relaxes the criteria for declaring the problem non-convex.

A problem is declared non-convex if negative (positive) pivot elements are detected in the Cholesky factor of a matrix which is required to be PSD (NSD). This parameter controls how much this non-negativity requirement may be violated.

If d_i is the pivot element for column i , then the matrix Q is considered to not be PSD if:

$$d_i \leq -|Q_{ii}|\text{check_convexity_rel_tol}$$

Default 1e-10

Accepted [0; +inf]

Groups *Interior-point method*

`dparam.data_sym_mat_tol`

Absolute zero tolerance for elements in symmetric 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.

Default 1.0e-12

Accepted [1.0e-16; 1.0e-6]

Groups *Data check*

`dparam.data_sym_mat_tol_huge`

An element in a symmetric matrix which is larger than this value in absolute size causes an error.

Default 1.0e20

Accepted [0.0; +inf]

Groups *Data check*

`dparam.data_sym_mat_tol_large`

An element in a symmetric matrix which is larger than this value in absolute size causes a warning message to be printed.

Default 1.0e10

Accepted [0.0; +inf]

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.

Default 1.0e-12

Accepted [1.0e-16; 1.0e-6]

Groups *Data check*

`dparam.data_tol_aij_huge`

An element in A which is larger than this value in absolute size causes an error.

Default 1.0e20

Accepted [0.0; +inf]

Groups *Data check*

`dparam.data_tol_aij_large`

An element in A which is larger than this value in absolute size causes a warning message to be printed.

Default 1.0e10

Accepted [0.0; +inf]

Groups *Data check*

`dparam.data_tol_bound_inf`

Any bound which in absolute value is greater than this parameter is considered infinite.

Default 1.0e16

Accepted [0.0; +inf]

Groups *Data check*

`dparam.data_tol_bound_wrn`

If a bound value is larger than this value in absolute size, then a warning message is issued.

Default 1.0e8

Accepted [0.0; +inf]

Groups *Data check*

`dparam.data_tol_c_huge`

An element in c which is larger than the value of this parameter in absolute terms is considered to be huge and generates an error.

Default 1.0e16

Accepted [0.0; +inf]

Groups *Data check*

`dparam.data_tol_cj_large`

An element in c which is larger than this value in absolute terms causes a warning message to be printed.

Default 1.0e8

Accepted [0.0; +inf]

Groups *Data check*

`dparam.data_tol_qij`

Absolute zero tolerance for elements in Q matrices.

Default 1.0e-16

Accepted [0.0; +inf]

Groups *Data check*

`dparam.data_tol_x`

Zero tolerance for constraints and variables i.e. if the distance between the lower and upper bound is less than this value, then the lower and upper bound is considered identical.

Default 1.0e-8

Accepted [0.0; +inf]

Groups *Data check*

`dparam.intpnt_co_tol_dfeas`

Dual feasibility tolerance used by the conic interior-point optimizer.

Default 1.0e-8

Accepted [0.0; 1.0]

Groups *Interior-point method, Termination criteria, Conic interior-point method*

See also [`dparam.intpnt_co_tol_near_rel`](#)

`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.

Default 1.0e-10

Accepted [0.0; 1.0]

Groups *Interior-point method, Termination criteria, Conic interior-point method*

`dparam.intpnt_co_tol_mu_red`

Relative complementarity gap feasibility tolerance used by the conic interior-point optimizer.

Default 1.0e-8

Accepted [0.0; 1.0]

Groups *Interior-point method, Termination criteria, 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.

Default 1000

Accepted [1.0; +inf]

Groups *Interior-point method, Termination criteria, Conic interior-point method*

`dparam.intpnt_co_tol_pfeas`

Primal feasibility tolerance used by the conic interior-point optimizer.

Default 1.0e-8

Accepted [0.0; 1.0]

Groups *Interior-point method, Termination criteria, Conic interior-point method*

See also [`dparam.intpnt_co_tol_near_rel`](#)

`dparam.intpnt_co_tol_rel_gap`

Relative gap termination tolerance used by the conic interior-point optimizer.

Default 1.0e-7

Accepted [0.0; 1.0]

Groups *Interior-point method, Termination criteria, Conic interior-point method*

See also [`dparam.intpnt_co_tol_near_rel`](#)

`dparam.intpnt_nl_merit_bal`

Controls if the complementarity and infeasibility is converging to zero at about equal rates.

Default 1.0e-4

Accepted [0.0; 0.99]

Groups *Interior-point method, Nonlinear convex method*

`dparam.intpnt_nl_tol_dfeas`

Dual feasibility tolerance used when a nonlinear model is solved.

Default 1.0e-8

Accepted [0.0; 1.0]

Groups *Interior-point method, Termination criteria, Nonlinear convex method*

`dparam.intpnt_nl_tol_mu_red`

Relative complementarity gap tolerance.

Default 1.0e-12

Accepted [0.0; 1.0]

Groups *Interior-point method, Termination criteria, 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.

Default 1000.0

Accepted [1.0; +inf]

Groups *Interior-point method, Termination criteria, Nonlinear convex method*

`dparam.intpnt_nl_tol_pfeas`

Primal feasibility tolerance used when a nonlinear model is solved.

Default 1.0e-8

Accepted [0.0; 1.0]

Groups *Interior-point method, Termination criteria, Nonlinear convex method*

`dparam.intpnt_nl_tol_rel_gap`

Relative gap termination tolerance for nonlinear problems.

Default 1.0e-6

Accepted [1.0e-14; +inf]

Groups *Termination criteria, Interior-point method, Nonlinear convex method*

`dparam.intpnt_nl_tol_rel_step`

Relative step size to the boundary for general nonlinear optimization problems.

Default 0.995

Accepted [1.0e-4; 0.9999999]

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..

Default 1.0e-8

Accepted [0.0; 1.0]

Groups *Interior-point method, Termination criteria*

See also `dparam.intpnt_qo_tol_near_rel`

`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.

Default 1.0e-10

Accepted [0.0; 1.0]

Groups *Interior-point method, Termination criteria*

`dparam.intpnt_qo_tol_mu_red`

Relative complementarity gap feasibility tolerance used when interior-point optimizer is applied to a quadratic optimization problem.

Default 1.0e-8

Accepted [0.0; 1.0]

Groups *Interior-point method, Termination criteria*

`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.

Default 1000

Accepted [1.0; +inf]

Groups *Interior-point method, Termination criteria*

`dparam.intpnt_qo_tol_pfeas`

Primal feasibility tolerance used when the interior-point optimizer is applied to a quadratic optimization problem.

Default 1.0e-8

Accepted [0.0; 1.0]

Groups *Interior-point method, Termination criteria*

See also `dparam.intpnt_qo_tol_near_rel`

`dparam.intpnt_qo_tol_rel_gap`

Relative gap termination tolerance used when the interior-point optimizer is applied to a quadratic optimization problem.

Default 1.0e-8

Accepted [0.0; 1.0]

Groups *Interior-point method, Termination criteria*

See also `dparam.intpnt_qo_tol_near_rel`

`dparam.intpnt_tol_dfeas`

Dual feasibility tolerance used for linear and quadratic optimization problems.

Default 1.0e-8

Accepted [0.0; 1.0]

Groups *Interior-point method, Termination criteria*

`dparam.intpnt_tol_dsafe`

Controls the initial dual starting point used by the interior-point optimizer. If the interior-point optimizer converges slowly and/or the constraint or variable bounds are very large, then it might be worthwhile to increase this value.

Default 1.0

Accepted [1.0e-4; +inf]

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.

Default 1.0e-10

Accepted [0.0; 1.0]

Groups *Interior-point method, Termination criteria, Nonlinear convex method*

`dparam.intpnt_tol_mu_red`

Relative complementarity gap tolerance.

Default 1.0e-16

Accepted [0.0; 1.0]

Groups *Interior-point method, Termination criteria*

`dparam.intpnt_tol_path`

Controls how close the interior-point optimizer follows the central path. A large value of this parameter means the central is followed very closely. On numerical unstable problems it may be worthwhile to increase this parameter.

Default 1.0e-8

Accepted [0.0; 0.9999]

Groups *Interior-point method*

`dparam.intpnt_tol_pfeas`

Primal feasibility tolerance used for linear and quadratic optimization problems.

Default 1.0e-8

Accepted [0.0; 1.0]

Groups *Interior-point method, Termination criteria*

`dparam.intpnt_tol_psafe`

Controls the initial primal starting point used by the interior-point optimizer. If the interior-point optimizer converges slowly and/or the constraint or variable bounds are very large, then it may be worthwhile to increase this value.

Default 1.0

Accepted [1.0e-4; +inf]

Groups *Interior-point method*

`dparam.intpnt_tol_rel_gap`

Relative gap termination tolerance.

Default 1.0e-8

Accepted [1.0e-14; +inf]

Groups *Termination criteria, Interior-point method*

`dparam.intpnt_tol_rel_step`

Relative step size to the boundary for linear and quadratic optimization problems.

Default 0.9999

Accepted [1.0e-4; 0.999999]

Groups *Interior-point method*

`dparam.intpnt_tol_step_size`

Minimal step size tolerance. If the step size falls below the value of this parameter, then the interior-point optimizer assumes that it is stalled. In other words the interior-point optimizer does not make any progress and therefore it is better stop.

Default 1.0e-6

Accepted [0.0; 1.0]

Groups *Interior-point method*

dparam.lower_obj_cut

If either a primal or dual feasible solution is found proving that the optimal objective value is outside, the interval [*dparam.lower_obj_cut*, *dparam.upper_obj_cut*], then **MOSEK** is terminated.

Default -1.0e30

Accepted [-inf; +inf]

Groups *Termination criteria*

See also *dparam.lower_obj_cut_finite_trh*

dparam.lower_obj_cut_finite_trh

If the lower objective cut is less than the value of this parameter value, then the lower objective cut i.e. *dparam.lower_obj_cut* is treated as $-\infty$.

Default -0.5e30

Accepted [-inf; +inf]

Groups *Termination criteria*

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.

Default -1.0

Accepted [-inf; +inf]

Groups *Mixed-integer optimization, Termination criteria*

See also *iparam.mio_max_num_relaxs*, *iparam.mio_max_num_branches*,
dparam.mio_near_tol_abs_gap, *dparam.mio_near_tol_rel_gap*

dparam.mio_max_time

This parameter limits the maximum time spent by the mixed-integer optimizer. A negative number means infinity.

Default -1.0

Accepted [-inf; +inf]

Groups *Mixed-integer optimization, Termination criteria*

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.

Default 0.0

Accepted [0.0; +inf]

Groups *Mixed-integer optimization*

See also *dparam.mio_disable_term_time*

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.

Default 1.0e-3

Accepted [0.0; +inf]

Groups *Mixed-integer optimization, Termination criteria*

See also [dparam.mio_disable_term_time](#)

dparam.mio_rel_gap_const

This value is used to compute the relative gap for the solution to an integer optimization problem.

Default 1.0e-10

Accepted [1.0e-15; +inf]

Groups *Mixed-integer optimization, Termination criteria*

dparam.mio_tol_abs_gap

Absolute optimality tolerance employed by the mixed-integer optimizer.

Default 0.0

Accepted [0.0; +inf]

Groups *Mixed-integer optimization*

dparam.mio_tol_abs_relax_int

Absolute integer feasibility tolerance. If the distance to the nearest integer is less than this tolerance then an integer constraint is assumed to be satisfied.

Default 1.0e-5

Accepted [1e-9; +inf]

Groups *Mixed-integer optimization*

dparam.mio_tol_feas

Feasibility tolerance for mixed integer solver.

Default 1.0e-6

Accepted [1e-9; 1e-3]

Groups *Mixed-integer optimization*

dparam.mio_tol_rel_dual_bound_improvement

If the relative improvement of the dual bound is smaller than this value, the solver will terminate the root cut generation. A value of 0.0 means that the value is selected automatically.

Default 0.0

Accepted [0.0; 1.0]

Groups *Mixed-integer optimization*

dparam.mio_tol_rel_gap

Relative optimality tolerance employed by the mixed-integer optimizer.

Default 1.0e-4

Accepted [0.0; +inf]

Groups *Mixed-integer optimization, Termination criteria*

dparam.optimizer_max_time

Maximum amount of time the optimizer is allowed to spent on the optimization. A negative number means infinity.

Default -1.0

Accepted $[-\text{inf}; +\text{inf}]$

Groups *Termination criteria*

dparam.presolve_tol_abs_lindep

Absolute tolerance employed by the linear dependency checker.

Default 1.0e-6

Accepted $[0.0; +\text{inf}]$

Groups *Presolve*

dparam.presolve_tol_aj

Absolute zero tolerance employed for a_{ij} in the presolve.

Default 1.0e-12

Accepted $[1.0\text{e-}15; +\text{inf}]$

Groups *Presolve*

dparam.presolve_tol_rel_lindep

Relative tolerance employed by the linear dependency checker.

Default 1.0e-10

Accepted $[0.0; +\text{inf}]$

Groups *Presolve*

dparam.presolve_tol_s

Absolute zero tolerance employed for s_i in the presolve.

Default 1.0e-8

Accepted $[0.0; +\text{inf}]$

Groups *Presolve*

dparam.presolve_tol_x

Absolute zero tolerance employed for x_j in the presolve.

Default 1.0e-8

Accepted $[0.0; +\text{inf}]$

Groups *Presolve*

dparam.qcqp_reformulate_rel_drop_tol

This parameter determines when columns are dropped in incomplete Cholesky factorization during reformulation of quadratic problems.

Default 1e-15

Accepted $[0; +\text{inf}]$

Groups *Interior-point method*

dparam.semidefinite_tol_approx

Tolerance to define a matrix to be positive semidefinite.

Default 1.0e-10

Accepted $[1.0\text{e-}15; +\text{inf}]$

Groups *Data check*

dparam.sim_lu_tol_rel_piv

Relative pivot tolerance employed when computing the LU factorization of the basis in the simplex optimizers and in the basis identification procedure.

A value closer to 1.0 generally improves numerical stability but typically also implies an increase in the computational work.

Default 0.01**Accepted** [1.0e-6; 0.999999]**Groups** *Basis identification, Simplex optimizer*`dparam.simplex_abs_tol_piv`

Absolute pivot tolerance employed by the simplex optimizers.

Default 1.0e-7**Accepted** [1.0e-12; +inf]**Groups** *Simplex optimizer*`dparam.upper_obj_cut`If either a primal or dual feasible solution is found proving that the optimal objective value is outside, the interval [`dparam.lower_obj_cut`, `dparam.upper_obj_cut`], then **MOSEK** is terminated.**Default** 1.0e30**Accepted** [-inf; +inf]**Groups** *Termination criteria***See also** `dparam.upper_obj_cut_finite_trh``dparam.upper_obj_cut_finite_trh`If the upper objective cut is greater than the value of this parameter, then the upper objective cut `dparam.upper_obj_cut` is treated as ∞ .**Default** 0.5e30**Accepted** [-inf; +inf]**Groups** *Termination criteria*

18.7.2 Integer parameters

`iparam`

The enumeration type containing all integer parameters.

`iparam.ana_sol_basis`

Controls whether the basis matrix is analyzed in solution analyzer.

Default *on***Accepted** *on, off* (see *onoffkey*)**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.**Default** *off***Accepted** *on, off* (see *onoffkey*)**Groups** *Analysis*`iparam.auto_sort_a_before_opt`Controls whether the elements in each column of *A* are sorted before an optimization is performed. This is not required but makes the optimization more deterministic.**Default** *off***Accepted** *on, off* (see *onoffkey*)

Groups *Debugging***iparam.auto_update_sol_info**

Controls whether the solution information items are automatically updated after an optimization is performed.

Default *off***Accepted** *on, off* (see *onoffkey*)**Groups** *Overall system***iparam.basis_solve_use_plus_one**

If a slack variable is in the basis, then the corresponding column in the basis is a unit vector with -1 in the right position. However, if this parameter is set to *onoffkey.on*, -1 is replaced by 1.

This has significance for the results returned by the *Task.solvewithbasis* function.

Default *off***Accepted** *on, off* (see *onoffkey*)**Groups** *Simplex optimizer***iparam.bi_clean_optimizer**

Controls which simplex optimizer is used in the clean-up phase.

Default *free***Accepted** *free, intpnt, conic, primal_simplex, dual_simplex, free_simplex, mixed_int* (see *optimizertype*)**Groups** *Basis identification, Overall solver***iparam.bi_ignore_max_iter**

If the parameter *iparam.intpnt_basis* has the value *basindtype.no_error* and the interior-point optimizer has terminated due to maximum number of iterations, then basis identification is performed if this parameter has the value *onoffkey.on*.

Default *off***Accepted** *on, off* (see *onoffkey*)**Groups** *Interior-point method, Basis identification***iparam.bi_ignore_num_error**

If the parameter *iparam.intpnt_basis* has the value *basindtype.no_error* and the interior-point optimizer has terminated due to a numerical problem, then basis identification is performed if this parameter has the value *onoffkey.on*.

Default *off***Accepted** *on, off* (see *onoffkey*)**Groups** *Interior-point method, Basis identification***iparam.bi_max_iterations**

Controls the maximum number of simplex iterations allowed to optimize a basis after the basis identification.

Default 1000000**Accepted** [0; +inf]**Groups** *Basis identification, Termination criteria***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.

Default *on*

Accepted *on, off* (see *onoffkey*)

Groups *License manager*

iparam.check_convexity

Specify the level of convexity check on quadratic problems.

Default *full*

Accepted *none, simple, full* (see *checkconvexitytype*)

Groups *Data check, Nonlinear convex method*

iparam.compress_statfile

Control compression of stat files.

Default *on*

Accepted *on, off* (see *onoffkey*)

iparam.infeas_generic_names

Controls whether generic names are used when an infeasible subproblem is created.

Default *off*

Accepted *on, off* (see *onoffkey*)

Groups *Infeasibility report*

iparam.infeas_prefer_primal

If both certificates of primal and dual infeasibility are supplied then only the primal is used when this option is turned on.

Default *on*

Accepted *on, off* (see *onoffkey*)

Groups *Overall solver*

iparam.infeas_report_auto

Controls whether an infeasibility report is automatically produced after the optimization if the problem is primal or dual infeasible.

Default *off*

Accepted *on, off* (see *onoffkey*)

Groups *Data input/output, Solution input/output*

iparam.infeas_report_level

Controls the amount of information presented in an infeasibility report. Higher values imply more information.

Default *1*

Accepted *[0; +inf]*

Groups *Infeasibility report, Output information*

iparam.intpnt_basis

Controls whether the interior-point optimizer also computes an optimal basis.

Default *always*

Accepted *never, always, no_error, if_feasible, reserved* (see *basindtype*)

Groups *Interior-point method, Basis identification*

See also *iparam.bi_ignore_max_iter*, *iparam.bi_ignore_num_error*,
iparam.bi_max_iterations, *iparam.bi_clean_optimizer*

`iparam.intpnt_diff_step`

Controls whether different step sizes are allowed in the primal and dual space.

Default *on*

Accepted

- *on*: Different step sizes are allowed.
- *off*: Different step sizes are not allowed.

Groups *Interior-point method*

`iparam.intpnt_hotstart`

Currently not in use.

Default *none*

Accepted *none, primal, dual, primal_dual* (see *intpnt_hotstart*)

Groups *Interior-point method*

`iparam.intpnt_max_iterations`

Controls the maximum number of iterations allowed in the interior-point optimizer.

Default 400

Accepted [0; +inf]

Groups *Interior-point method, Termination criteria*

`iparam.intpnt_max_num_cor`

Controls the maximum number of correctors allowed by the multiple corrector procedure. A negative value means that **MOSEK** is making the choice.

Default -1

Accepted [-1; +inf]

Groups *Interior-point method*

`iparam.intpnt_max_num_refinement_steps`

Maximum number of steps to be used by the iterative refinement of the search direction. A negative value implies that the optimizer chooses the maximum number of iterative refinement steps.

Default -1

Accepted [-inf; +inf]

Groups *Interior-point method*

`iparam.intpnt_multi_thread`

Controls whether the interior-point optimizers are allowed to employ multiple threads if more threads is available.

Default *on*

Accepted *on, off* (see *onoffkey*)

Groups *Overall system*

`iparam.intpnt_off_col_trh`

Controls how many offending columns are detected in the Jacobian of the constraint matrix.

0	no detection
1	aggressive detection
> 1	higher values mean less aggressive detection

Default 40

Accepted [0; +inf]

Groups *Interior-point method*

`iparam.intpnt_order_method`

Controls the ordering strategy used by the interior-point optimizer when factorizing the Newton equation system.

Default *free*

Accepted *free, appminloc, experimental, try_graphpar, force_graphpar, none*
(see *orderingtype*)

Groups *Interior-point method*

`iparam.intpnt_regularization_use`

Controls whether regularization is allowed.

Default *on*

Accepted *on, off* (see *onoffkey*)

Groups *Interior-point method*

`iparam.intpnt_scaling`

Controls how the problem is scaled before the interior-point optimizer is used.

Default *free*

Accepted *free, none, moderate, aggressive* (see *scalingtype*)

Groups *Interior-point method*

`iparam.intpnt_solve_form`

Controls whether the primal or the dual problem is solved.

Default *free*

Accepted *free, primal, dual* (see *solveform*)

Groups *Interior-point method*

`iparam.intpnt_starting_point`

Starting point used by the interior-point optimizer.

Default *free*

Accepted *free, guess, constant, satisfy_bounds* (see *startpointtype*)

Groups *Interior-point method*

`iparam.license_debug`

This option is used to turn on debugging of the license manager.

Default *off*

Accepted *on, off* (see *onoffkey*)

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.

Default 100

Accepted [0; 1000000]

Groups *License manager*

`iparam.license_suppress_expire_wrns`

Controls whether license features expire warnings are suppressed.

Default *off*

Accepted *on, off* (see *onoffkey*)

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.

Default 7

Accepted [0; +inf]

Groups *License manager, Output information*

`iparam.license_wait`

If all licenses are in use **MOSEK** returns with an error code. However, by turning on this parameter **MOSEK** will wait for an available license.

Default *off*

Accepted *on, off* (see *onoffkey*)

Groups *Overall solver, Overall system, License manager*

`iparam.log`

Controls the amount of log information. The value 0 implies that all log information is suppressed. A higher level implies that more information is logged.

Please note that if a task is employed to solve a sequence of optimization problems the value of this parameter is reduced by the value of *iparam.log_cut_second_opt* for the second and any subsequent optimizations.

Default 10

Accepted [0; +inf]

Groups *Output information, Logging*

See also *iparam.log_cut_second_opt*

`iparam.log_ana_pro`

Controls amount of output from the problem analyzer.

Default 1

Accepted [0; +inf]

Groups *Analysis, Logging*

`iparam.log_bi`

Controls the amount of output printed by the basis identification procedure. A higher level implies that more information is logged.

Default 1

Accepted [0; +inf]

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 callback function is called.

Default 2500

Accepted [0; +inf]

Groups *Basis identification, Output information, Logging*

iparam.log_check_convexity

Controls logging in convexity check on quadratic problems. Set to a positive value to turn logging on. If a quadratic coefficient matrix is found to violate the requirement of PSD (NSD) then a list of negative (positive) pivot elements is printed. The absolute value of the pivot elements is also shown.

Default 0

Accepted [0; +inf]

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.

Default 1

Accepted [0; +inf]

Groups *Output information, Logging*

See also *iparam.log, iparam.log_intpnt, iparam.log_mio, iparam.log_sim*

iparam.log_expand

Controls the amount of logging when a data item such as the maximum number constraints is expanded.

Default 0

Accepted [0; +inf]

Groups *Output information, Logging*

iparam.log_feas_repair

Controls the amount of output printed when performing feasibility repair. A value higher than one means extensive logging.

Default 1

Accepted [0; +inf]

Groups *Output information, Logging*

iparam.log_file

If turned on, then some log info is printed when a file is written or read.

Default 1

Accepted [0; +inf]

Groups *Data input/output, Output information, Logging*

iparam.log_infeas_ana

Controls amount of output printed by the infeasibility analyzer procedures. A higher level implies that more information is logged.

Default 1

Accepted [0; +inf]

Groups *Infeasibility report, Output information, Logging*

iparam.log_intpnt

Controls amount of output printed by the interior-point optimizer. A higher level implies that more information is logged.

Default 1

Accepted [0; +inf]

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.

Default 4

Accepted [0; +inf]

Groups *Mixed-integer optimization, Output information, Logging*

`iparam.log_mio_freq`

Controls how frequent the mixed-integer optimizer prints the log line. It will print line every time *iparam.log_mio_freq* relaxations have been solved.

Default 10

Accepted [-inf; +inf]

Groups *Mixed-integer optimization, Output information, Logging*

`iparam.log_order`

If turned on, then factor lines are added to the log.

Default 1

Accepted [0; +inf]

Groups *Output information, Logging*

`iparam.log_presolve`

Controls amount of output printed by the presolve procedure. A higher level implies that more information is logged.

Default 1

Accepted [0; +inf]

Groups *Logging*

`iparam.log_response`

Controls amount of output printed when response codes are reported. A higher level implies that more information is logged.

Default 0

Accepted [0; +inf]

Groups *Output information, Logging*

`iparam.log_sensitivity`

Controls the amount of logging during the sensitivity analysis.

0.Means no logging information is produced.

1.Timing information is printed.

2.Sensitivity results are printed.

Default 1

Accepted [0; +inf]

Groups *Output information, Logging*

`iparam.log_sensitivity_opt`

Controls the amount of logging from the optimizers employed during the sensitivity analysis. 0 means no logging information is produced.

Default 0

Accepted [0; +inf]

Groups *Output information, Logging*

`iparam.log_sim`

Controls amount of output printed by the simplex optimizer. A higher level implies that more information is logged.

Default 4

Accepted [0; +inf]

Groups *Simplex optimizer, Output information, Logging*

`iparam.log_sim_freq`

Controls how frequent the simplex optimizer outputs information about the optimization and how frequent the user-defined callback function is called.

Default 1000

Accepted [0; +inf]

Groups *Simplex optimizer, Output information, Logging*

`iparam.log_sim_minor`

Currently not in use.

Default 1

Accepted [0; +inf]

Groups *Simplex optimizer, Output information*

`iparam.log_storage`

When turned on, **MOSEK** prints messages regarding the storage usage and allocation.

Default 0

Accepted [0; +inf]

Groups *Output information, Overall 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.

Default 10

Accepted [-inf; +inf]

Groups *Output information*

`iparam.mio_branch_dir`

Controls whether the mixed-integer optimizer is branching up or down by default.

Default *free*

Accepted *free, up, down, near, far, root_lp, guided, pseudocost* (see *branchdir*)

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.

Default *off*

Accepted *on, off* (see *onoffkey*)

Groups *Mixed-integer optimization*

`iparam.mio_cut_clique`

Controls whether clique cuts should be generated.

Default *on*

Accepted

- *on*: Turns generation of this cut class on.
- *off*: Turns generation of this cut class off.

Groups *Mixed-integer optimization*

`iparam.mio_cut_cmir`

Controls whether mixed integer rounding cuts should be generated.

Default *on*

Accepted

- *on*: Turns generation of this cut class on.
- *off*: Turns generation of this cut class off.

Groups *Mixed-integer optimization*

`iparam.mio_cut_gmi`

Controls whether GMI cuts should be generated.

Default *on*

Accepted

- *on*: Turns generation of this cut class on.
- *off*: Turns generation of this cut class off.

Groups *Mixed-integer optimization*

`iparam.mio_cut_implied_bound`

Controls whether implied bound cuts should be generated.

Default *off*

Accepted

- *on*: Turns generation of this cut class on.
- *off*: Turns generation of this cut class off.

Groups *Mixed-integer optimization*

`iparam.mio_cut_knapsack_cover`

Controls whether knapsack cover cuts should be generated.

Default *off*

Accepted

- *on*: Turns generation of this cut class on.
- *off*: Turns generation of this cut class 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

Default -1

Accepted [-1; +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.

Default -1

Accepted [-inf; +inf]

Groups *Mixed-integer optimization*

`iparam.mio_max_num_branches`

Maximum number of branches allowed during the branch and bound search. A negative value means infinite.

Default -1

Accepted [-inf; +inf]

Groups *Mixed-integer optimization, Termination criteria*

See also `dparam.mio_disable_term_time`

`iparam.mio_max_num_relaxs`

Maximum number of relaxations allowed during the branch and bound search. A negative value means infinite.

Default -1

Accepted [-inf; +inf]

Groups *Mixed-integer optimization*

See also `dparam.mio_disable_term_time`

`iparam.mio_max_num_solutions`

The mixed-integer optimizer can be terminated after a certain number of different feasible solutions has been located. If this parameter has the value $n > 0$, then the mixed-integer optimizer will be terminated when n feasible solutions have been located.

Default -1

Accepted [-inf; +inf]

Groups *Mixed-integer optimization, Termination criteria*

See also `dparam.mio_disable_term_time`

`iparam.mio_mode`

Controls whether the optimizer includes the integer restrictions when solving a (mixed) integer optimization problem.

Default *satisfied*

Accepted *ignored, satisfied* (see *miomode*)

Groups *Overall solver*

`iparam.mio_mt_user_cb`

If true user callbacks are called from each thread used by mixed-integer optimizer. Otherwise it is only called from a single thread.

Default *off*

Accepted *on, off* (see *onoffkey*)

Groups *Overall system***iparam.mio_node_optimizer**

Controls which optimizer is employed at the non-root nodes in the mixed-integer optimizer.

Default *free***Accepted** *free, intpnt, conic, primal_simplex, dual_simplex, free_simplex, mixed_int* (see *optimizertype*)**Groups** *Mixed-integer optimization***iparam.mio_node_selection**

Controls the node selection strategy employed by the mixed-integer optimizer.

Default *free***Accepted** *free, first, best, worst, hybrid, pseudo* (see *mionodeseltype*)**Groups** *Mixed-integer optimization***iparam.mio_perspective_reformulate**

Enables or disables perspective reformulation in presolve.

Default *on***Accepted** *on, off* (see *onoffkey*)**Groups** *Mixed-integer optimization***iparam.mio_probing_level**

Controls the amount of probing employed by the mixed-integer optimizer in presolve.

-1. The optimizer chooses the level of probing employed

0. Probing is disabled

1. A low amount of probing is employed

2. A medium amount of probing is employed

3. A high amount of probing is employed

Default *-1***Accepted** *[-1; 3]***Groups** *Mixed-integer optimization***iparam.mio_rins_max_nodes**

Controls the maximum number of nodes allowed in each call to the RINS heuristic. The default value of -1 means that the value is determined automatically. A value of zero turns off the heuristic.

Default *-1***Accepted** *[-1; +inf]***Groups** *Mixed-integer optimization***iparam.mio_root_optimizer**

Controls which optimizer is employed at the root node in the mixed-integer optimizer.

Default *free***Accepted** *free, intpnt, conic, primal_simplex, dual_simplex, free_simplex, mixed_int* (see *optimizertype*)**Groups** *Mixed-integer optimization***iparam.mio_root_repeat_presolve_level**

Controls whether presolve can be repeated at root node.

• -1 The optimizer chooses whether presolve is repeated

- 0 Never repeat presolve
- 1 Always repeat presolve

Default -1

Accepted [-1; 1]

Groups *Mixed-integer optimization*

`iparam.mio_vb_detection_level`

Controls how much effort is put into detecting variable bounds.

-1. The optimizer chooses

0.No variable bounds are detected

1.Only detect variable bounds that are directly represented in the problem

2.Detect variable bounds in probing

Default -1

Accepted [-1; +2]

Groups *Mixed-integer optimization*

`iparam.mt_spincount`

Set the number of iterations to spin before sleeping.

Default 0

Accepted [0; 1000000000]

Groups *Overall system*

`iparam.num_threads`

Controls the number of threads employed by the optimizer. If set to 0 the number of threads used will be equal to the number of cores detected on the machine.

Default 0

Accepted [0; +inf]

Groups *Overall system*

`iparam.opf_max_terms_per_line`

The maximum number of terms (linear and quadratic) per line when an OPF file is written.

Default 5

Accepted [0; +inf]

Groups *Data input/output*

`iparam.opf_write_header`

Write a text header with date and **MOSEK** version in an OPF file.

Default *on*

Accepted *on*, *off* (see *onoffkey*)

Groups *Data input/output*

`iparam.opf_write_hints`

Write a hint section with problem dimensions in the beginning of an OPF file.

Default *on*

Accepted *on*, *off* (see *onoffkey*)

Groups *Data input/output*

`iparam.opf_write_parameters`

Write a parameter section in an OPF file.

Default *off*

Accepted *on, off* (see *onoffkey*)

Groups *Data input/output*

`iparam.opf_write_problem`

Write objective, constraints, bounds etc. to an OPF file.

Default *on*

Accepted *on, off* (see *onoffkey*)

Groups *Data input/output*

`iparam.opf_write_sol_bas`

If *iparam.opf_write_solutions* is *onoffkey.on* and a basic solution is defined, include the basic solution in OPF files.

Default *on*

Accepted *on, off* (see *onoffkey*)

Groups *Data input/output*

`iparam.opf_write_sol_itg`

If *iparam.opf_write_solutions* is *onoffkey.on* and an integer solution is defined, write the integer solution in OPF files.

Default *on*

Accepted *on, off* (see *onoffkey*)

Groups *Data input/output*

`iparam.opf_write_sol_itr`

If *iparam.opf_write_solutions* is *onoffkey.on* and an interior solution is defined, write the interior solution in OPF files.

Default *on*

Accepted *on, off* (see *onoffkey*)

Groups *Data input/output*

`iparam.opf_write_solutions`

Enable inclusion of solutions in the OPF files.

Default *off*

Accepted *on, off* (see *onoffkey*)

Groups *Data input/output*

`iparam.optimizer`

The parameter controls which optimizer is used to optimize the task.

Default *free*

Accepted *free, intpnt, conic, primal_simplex, dual_simplex, free_simplex, mixed_int* (see *optimizertype*)

Groups *Overall solver*

`iparam.param_read_case_name`

If turned on, then names in the parameter file are case sensitive.

Default *on*

Accepted *on, off* (see *onoffkey*)

Groups *Data input/output*

`iparam.param_read_ign_error`

If turned on, then errors in parameter settings is ignored.

Default *off*

Accepted *on, off* (see *onoffkey*)

Groups *Data input/output*

`iparam.presolve_eliminator_max_fill`

Controls the maximum amount of fill-in that can be created by one pivot in the elimination phase of the presolve. A negative value means the parameter value is selected automatically.

Default -1

Accepted [-inf; +inf]

Groups *Presolve*

`iparam.presolve_eliminator_max_num_tries`

Control the maximum number of times the eliminator is tried. A negative value implies **MOSEK** decides.

Default -1

Accepted [-inf; +inf]

Groups *Presolve*

`iparam.presolve_level`

Currently not used.

Default -1

Accepted [-inf; +inf]

Groups *Overall solver, Presolve*

`iparam.presolve_lindep_abs_work_trh`

The linear dependency check is potentially computationally expensive.

Default 100

Accepted [-inf; +inf]

Groups *Presolve*

`iparam.presolve_lindep_rel_work_trh`

The linear dependency check is potentially computationally expensive.

Default 100

Accepted [-inf; +inf]

Groups *Presolve*

`iparam.presolve_lindep_use`

Controls whether the linear constraints are checked for linear dependencies.

Default *on*

Accepted

- *on*: Turns the linear dependency check on.
- *off*: Turns the linear dependency check off.

Groups *Presolve*

iparam.presolve_max_num_reductions

Controls the maximum number of reductions performed by the presolve. The value of the parameter is normally only changed in connection with debugging. A negative value implies that an infinite number of reductions are allowed.

Default -1

Accepted [-inf; +inf]

Groups *Overall solver, Presolve*

iparam.presolve_use

Controls whether the presolve is applied to a problem before it is optimized.

Default *free*

Accepted *off, on, free* (see *presolvemode*)

Groups *Overall solver, Presolve*

iparam.primal_repair_optimizer

Controls which optimizer that is used to find the optimal repair.

Default *free*

Accepted *free, intpnt, conic, primal_simplex, dual_simplex, free_simplex, mixed_int* (see *optimizertype*)

Groups *Overall solver*

iparam.read_data_compressed

If this option is turned on, it is assumed that the data file is compressed.

Default *free*

Accepted *none, free, gzip* (see *compresstype*)

Groups *Data input/output*

iparam.read_data_format

Format of the data file to be read.

Default *extension*

Accepted *extension, mps, lp, op, xml, free_mps, task, cb, json_task* (see *dataformat*)

Groups *Data input/output*

iparam.read_debug

Turns on additional debugging information when reading files.

Default *off*

Accepted *on, off* (see *onoffkey*)

Groups *Data input/output*

iparam.read_keep_free_con

Controls whether the free constraints are included in the problem.

Default *off*

Accepted

- *on*: The free constraints are kept.
- *off*: The free constraints are discarded.

Groups *Data input/output*

`iparam.read_lp_drop_new_vars_in_bou`

If this option is turned on, **MOSEK** will drop variables that are defined for the first time in the bounds section.

Default *off*

Accepted *on, off* (see *onoffkey*)

Groups *Data input/output*

`iparam.read_lp_quoted_names`

If a name is in quotes when reading an LP file, the quotes will be removed.

Default *on*

Accepted *on, off* (see *onoffkey*)

Groups *Data input/output*

`iparam.read_mps_format`

Controls how strictly the MPS file reader interprets the MPS format.

Default *free*

Accepted *strict, relaxed, free, cplex* (see *mpsformat*)

Groups *Data input/output*

`iparam.read_mps_width`

Controls the maximal number of characters allowed in one line of the MPS file.

Default 1024

Accepted [80; +inf]

Groups *Data input/output*

`iparam.read_task_ignore_param`

Controls whether **MOSEK** should ignore the parameter setting defined in the task file and use the default parameter setting instead.

Default *off*

Accepted *on, off* (see *onoffkey*)

Groups *Data input/output*

`iparam.remove_unused_solutions`

Removes unused solutions before the optimization is performed.

Default *off*

Accepted *on, off* (see *onoffkey*)

Groups *Overall system*

`iparam.sensitivity_all`

If set to *onoffkey.on*, then *Task.sensitivityreport* analyzes all bounds and variables instead of reading a specification from the file.

Default *off*

Accepted *on, off* (see *onoffkey*)

Groups *Overall solver*

`iparam.sensitivity_optimizer`

Controls which optimizer is used for optimal partition sensitivity analysis.

Default *free_simplex*

Accepted *free, intpnt, conic, primal_simplex, dual_simplex, free_simplex, mixed_int* (see *optimizertype*)

Groups *Overall solver, Simplex optimizer*

`iparam.sensitivity_type`

Controls which type of sensitivity analysis is to be performed.

Default *basis*

Accepted *basis, optimal_partition* (see *sensitivitytype*)

Groups *Overall solver*

`iparam.sim_basis_factor_use`

Controls whether an LU factorization of the basis is used in a hot-start. Forcing a refactorization sometimes improves the stability of the simplex optimizers, but in most cases there is a performance penalty.

Default *on*

Accepted *on, off* (see *onoffkey*)

Groups *Simplex optimizer*

`iparam.sim_degen`

Controls how aggressively degeneration is handled.

Default *free*

Accepted *none, free, aggressive, moderate, minimum* (see *simdegen*)

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 $(100 - x) \bmod f_v$ entries, where f_v is the number of fixed variables.

Default 90

Accepted $[0; +\infty]$

Groups *Dual simplex*

`iparam.sim_dual_phaseone_method`

An experimental feature.

Default 0

Accepted $[0; 10]$

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.

Default 50

Accepted $[0; 100]$

Groups *Dual simplex*

`iparam.sim_dual_selection`

Controls the choice of the incoming variable, known as the selection strategy, in the dual simplex optimizer.

Default *free*

Accepted *free, full, ase, devex, se, partial* (see *simseltype*)

Groups *Dual simplex*

iparam.sim_exploit_dupvec

Controls if the simplex optimizers are allowed to exploit duplicated columns.

Default *off*

Accepted *on, off, free* (see *simdupvec*)

Groups *Simplex optimizer*

iparam.sim_hotstart

Controls the type of hot-start that the simplex optimizer perform.

Default *free*

Accepted *none, free, status_keys* (see *simhotstart*)

Groups *Simplex optimizer*

iparam.sim_hotstart_lu

Determines if the simplex optimizer should exploit the initial factorization.

Default *on*

Accepted

- *on*: Factorization is reused if possible.
- *off*: Factorization is recomputed.

Groups *Simplex optimizer*

iparam.sim_max_iterations

Maximum number of iterations that can be used by a simplex optimizer.

Default 10000000

Accepted [0; +inf]

Groups *Simplex optimizer, Termination criteria*

iparam.sim_max_num_setbacks

Controls how many set-backs are allowed within a simplex optimizer. A set-back is an event where the optimizer moves in the wrong direction. This is impossible in theory but may happen due to numerical problems.

Default 250

Accepted [0; +inf]

Groups *Simplex optimizer*

iparam.sim_non_singular

Controls if the simplex optimizer ensures a non-singular basis, if possible.

Default *on*

Accepted *on, off* (see *onoffkey*)

Groups *Simplex optimizer*

iparam.sim_primal_crash

Controls whether crashing is performed in the primal simplex optimizer.

In general, if a basis consists of more than (100-this parameter value)% fixed variables, then a crash will be performed.

Default 90

Accepted [0; +inf]

Groups *Primal simplex***iparam.sim_primal_phaseone_method**

An experimental feature.

Default 0**Accepted** [0; 10]**Groups** *Simplex optimizer***iparam.sim_primal_restrict_selection**

The primal simplex optimizer can use a so-called restricted selection/pricing strategy to choose the outgoing variable. Hence, if restricted selection is applied, then the primal simplex optimizer first choose a subset of all the potential incoming variables. Next, for some time it will choose the incoming variable only among the subset. From time to time the subset is redefined.

A larger value of this parameter implies that the optimizer will be more aggressive in its restriction strategy, i.e. a value of 0 implies that the restriction strategy is not applied at all.

Default 50**Accepted** [0; 100]**Groups** *Primal simplex***iparam.sim_primal_selection**

Controls the choice of the incoming variable, known as the selection strategy, in the primal simplex optimizer.

Default *free***Accepted** *free, full, ase, devex, se, partial* (see *simseltype*)**Groups** *Primal simplex***iparam.sim_refactor_freq**

Controls how frequent the basis is refactorized. The value 0 means that the optimizer determines the best point of refactorization.

It is strongly recommended NOT to change this parameter.

Default 0**Accepted** [0; +inf]**Groups** *Simplex optimizer***iparam.sim_reformulation**

Controls if the simplex optimizers are allowed to reformulate the problem.

Default *off***Accepted** *on, off, free, aggressive* (see *simreform*)**Groups** *Simplex optimizer***iparam.sim_save_lu**

Controls if the LU factorization stored should be replaced with the LU factorization corresponding to the initial basis.

Default *off***Accepted** *on, off* (see *onoffkey*)**Groups** *Simplex optimizer***iparam.sim_scaling**

Controls how much effort is used in scaling the problem before a simplex optimizer is used.

Default *free***Accepted** *free, none, moderate, aggressive* (see *scalingtype*)

Groups *Simplex optimizer***iparam.sim_scaling_method**

Controls how the problem is scaled before a simplex optimizer is used.

Default *pow2***Accepted** *pow2, free* (see *scalingmethod*)**Groups** *Simplex optimizer***iparam.sim_solve_form**

Controls whether the primal or the dual problem is solved by the primal-/dual-simplex optimizer.

Default *free***Accepted** *free, primal, dual* (see *solveform*)**Groups** *Simplex optimizer***iparam.sim_stability_priority**

Controls how high priority the numerical stability should be given.

Default 50**Accepted** [0; 100]**Groups** *Simplex optimizer***iparam.sim_switch_optimizer**

The simplex optimizer sometimes chooses to solve the dual problem instead of the primal problem. This implies that if you have chosen to use the dual simplex optimizer and the problem is dualized, then it actually makes sense to use the primal simplex optimizer instead. If this parameter is on and the problem is dualized and furthermore the simplex optimizer is chosen to be the primal (dual) one, then it is switched to the dual (primal).

Default *off***Accepted** *on, off* (see *onoffkey*)**Groups** *Simplex optimizer***iparam.sol_filter_keep_basic**

If turned on, then basic and super basic constraints and variables are written to the solution file independent of the filter setting.

Default *off***Accepted** *on, off* (see *onoffkey*)**Groups** *Solution input/output***iparam.sol_filter_keep_ranged**

If turned on, then ranged constraints and variables are written to the solution file independent of the filter setting.

Default *off***Accepted** *on, off* (see *onoffkey*)**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 must be specified. A negative value implies that no name contain blanks.

Default -1**Accepted** [-inf; +inf]**Groups** *Data input/output, Solution input/output*

`iparam.sol_read_width`

Controls the maximal acceptable width of line in the solutions when read by MOSEK.

Default 1024

Accepted [80; +inf]

Groups *Data input/output, Solution input/output*

`iparam.solution_callback`

Indicates whether solution callbacks will be performed during the optimization.

Default *off*

Accepted *on, off* (see *onoffkey*)

Groups *Progress callback, Overall solver*

`iparam.timing_level`

Controls the amount of timing performed inside MOSEK.

Default 1

Accepted [0; +inf]

Groups *Overall system*

`iparam.write_bas_constraints`

Controls whether the constraint section is written to the basic solution file.

Default *on*

Accepted *on, off* (see *onoffkey*)

Groups *Data input/output, Solution input/output*

`iparam.write_bas_head`

Controls whether the header section is written to the basic solution file.

Default *on*

Accepted *on, off* (see *onoffkey*)

Groups *Data input/output, Solution input/output*

`iparam.write_bas_variables`

Controls whether the variables section is written to the basic solution file.

Default *on*

Accepted *on, off* (see *onoffkey*)

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.

Default 0

Accepted [0; +inf]

Groups *Data input/output*

`iparam.write_data_format`

Controls the data format when a task is written using *Task.writedata*.

Default *extension*

Accepted *extension, mps, lp, op, xml, free_mps, task, cb, json_task* (see *dataformat*)

Groups *Data input/output*

iparam.write_data_param

If this option is turned on the parameter settings are written to the data file as parameters.

Default *off*

Accepted *on, off* (see *onoffkey*)

Groups *Data input/output*

iparam.write_free_con

Controls whether the free constraints are written to the data file.

Default *on*

Accepted

- *on*: The free constraints are written.
- *off*: The free constraints are discarded.

Groups *Data input/output*

iparam.write_generic_names

Controls whether the generic names or user-defined names are used in the data file.

Default *off*

Accepted

- *on*: Generic names are used.
- *off*: Generic names are not used.

Groups *Data input/output*

iparam.write_generic_names_io

Index origin used in generic names.

Default 1

Accepted [0; +inf]

Groups *Data input/output*

iparam.write_ignore_incompatible_items

Controls if the writer ignores incompatible problem items when writing files.

Default *off*

Accepted

- *on*: Ignore items that cannot be written to the current output file format.
- *off*: Produce an error if the problem contains items that cannot be written to the current output file format.

Groups *Data input/output*

iparam.write_int_constraints

Controls whether the constraint section is written to the integer solution file.

Default *on*

Accepted *on, off* (see *onoffkey*)

Groups *Data input/output, Solution input/output*

iparam.write_int_head

Controls whether the header section is written to the integer solution file.

Default *on*

Accepted *on, off* (see *onoffkey*)

Groups *Data input/output, Solution input/output*

`iparam.write_int_variables`

Controls whether the variables section is written to the integer solution file.

Default *on*

Accepted *on, off* (see *onoffkey*)

Groups *Data input/output, Solution input/output*

`iparam.write_lp_full_obj`

Write all variables, including the ones with 0-coefficients, in the objective.

Default *on*

Accepted *on, off* (see *onoffkey*)

Groups *Data input/output*

`iparam.write_lp_line_width`

Maximum width of line in an LP file written by **MOSEK**.

Default 80

Accepted [40; +inf]

Groups *Data input/output*

`iparam.write_lp_quoted_names`

If this option is turned on, then **MOSEK** will quote invalid LP names when writing an LP file.

Default *on*

Accepted *on, off* (see *onoffkey*)

Groups *Data input/output*

`iparam.write_lp_strict_format`

Controls whether LP output files satisfy the LP format strictly.

Default *off*

Accepted *on, off* (see *onoffkey*)

Groups *Data input/output*

`iparam.write_lp_terms_per_line`

Maximum number of terms on a single line in an LP file written by **MOSEK**. 0 means unlimited.

Default 10

Accepted [0; +inf]

Groups *Data input/output*

`iparam.write_mps_format`

Controls in which format the MPS is written.

Default *free*

Accepted *strict, relaxed, free, cplex* (see *mpsformat*)

Groups *Data input/output*

`iparam.write_mps_int`

Controls if marker records are written to the MPS file to indicate whether variables are integer restricted.

Default *on*

Accepted

- *on*: Marker records are written.
- *off*: Marker records are not written.

Groups *Data input/output***iparam.write_precision**

Controls the precision with which double numbers are printed in the MPS data file. In general it is not worthwhile to use a value higher than 15.

Default 15

Accepted [0; +inf]

Groups *Data input/output*

iparam.write_sol_barvariables

Controls whether the symmetric matrix variables section is written to the solution file.

Default *on*

Accepted *on, off* (see *onoffkey*)

Groups *Data input/output, Solution input/output*

iparam.write_sol_constraints

Controls whether the constraint section is written to the solution file.

Default *on*

Accepted *on, off* (see *onoffkey*)

Groups *Data input/output, Solution input/output*

iparam.write_sol_head

Controls whether the header section is written to the solution file.

Default *on*

Accepted *on, off* (see *onoffkey*)

Groups *Data input/output, Solution input/output*

iparam.write_sol_ignore_invalid_names

Even if the names are invalid MPS names, then they are employed when writing the solution file.

Default *off*

Accepted *on, off* (see *onoffkey*)

Groups *Data input/output, Solution input/output*

iparam.write_sol_variables

Controls whether the variables section is written to the solution file.

Default *on*

Accepted *on, off* (see *onoffkey*)

Groups *Data input/output, Solution input/output*

iparam.write_task_inc_sol

Controls whether the solutions are stored in the task file too.

Default *on*

Accepted *on, off* (see *onoffkey*)

Groups *Data input/output*

iparam.write_xml_mode

Controls if linear coefficients should be written by row or column when writing in the XML file format.

Default *row*

Accepted *row, col* (see *xmlwriteroutputtype*)

Groups *Data input/output*

18.7.3 String parameters

sparam

The enumeration type containing all string parameters.

sparam.bas_sol_file_name

Name of the bas solution file.

Accepted Any valid file name.

Groups *Data input/output, Solution input/output*

sparam.data_file_name

Data are read and written to this file.

Accepted Any valid file name.

Groups *Data input/output*

sparam.debug_file_name

MOSEK debug file.

Accepted Any valid file name.

Groups *Data input/output*

sparam.int_sol_file_name

Name of the int solution file.

Accepted Any valid file name.

Groups *Data input/output, Solution input/output*

sparam.itr_sol_file_name

Name of the itr solution file.

Accepted Any valid file name.

Groups *Data input/output, Solution input/output*

sparam.mio_debug_string

For internal debugging purposes.

Accepted 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.

Default

%%

Accepted Any valid string.

Groups *Data input/output*

sparam.param_read_file_name

Modifications to the parameter database is read from this file.

Accepted Any valid file name.

Groups *Data input/output*

sparam.param_write_file_name

The parameter database is written to this file.

Accepted 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 Any valid MPS name.

Groups *Data input/output*

`sparam.read_mps_obj_name`

Name of the free constraint used as objective function. An empty name means that the first constraint is used as objective function.

Accepted Any valid MPS name.

Groups *Data input/output*

`sparam.read_mps_ran_name`

Name of the RANGE vector used. An empty name means that the first RANGE vector is used.

Accepted Any valid MPS name.

Groups *Data input/output*

`sparam.read_mps_rhs_name`

Name of the RHS used. An empty name means that the first RHS vector is used.

Accepted Any valid MPS name.

Groups *Data input/output*

`sparam.remote_access_token`

An access token used to submit tasks to a remote **MOSEK** server. An access token is a random 32-byte string encoded in base64, i.e. it is a 44 character ASCII string.

Accepted Any valid string.

Groups *Overall system*

`sparam.sensitivity_file_name`

If defined *Task.sensitivityreport* reads this file as a sensitivity analysis data file specifying the type of analysis to be done.

Accepted Any valid string.

Groups *Data input/output*

`sparam.sensitivity_res_file_name`

If this is a nonempty string, then *Task.sensitivityreport* writes results to this file.

Accepted Any valid string.

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 $xc[i] \geq blc[i] + 0.5$ should be listed. An empty filter means that no filter is applied.

Accepted 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 $xc[i] \leq buc[i] - 0.5$ should be listed. An empty filter means that no filter is applied.

Accepted 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 $xx[j] \geq 0.5$ should be listed, whereas “+0.5” means that all constraints having $xx[j] \geq blx[j] + 0.5$ should be listed. An empty filter means no filter is applied.

Accepted Any valid filter.

Groups *Data input/output, Solution input/output*

`sparam.sol_filter_xx_upr`

A filter used to determine which variables should be listed in the solution file. A value of “0.5” means that all constraints having $xx[j] < 0.5$ should be printed, whereas “-0.5” means all constraints having $xx[j] \leq bux[j] - 0.5$ should be listed. An empty filter means no filter is applied.

Accepted Any valid file name.

Groups *Data input/output, Solution input/output*

`sparam.stat_file_name`

Statistics file name.

Accepted Any valid file name.

Groups *Data input/output*

`sparam.stat_key`

Key used when writing the summary file.

Accepted Any valid string.

Groups *Data input/output*

`sparam.stat_name`

Name used when writing the statistics file.

Accepted 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.

Default `xmskgen`

Accepted Any valid string.

Groups *Data input/output*

18.8 Response codes

- *Termination*
- *Warnings*
- *Errors*

`rescode`

The enumeration type containing all response codes.

18.8.1 Termination

`rescode.ok`

No error occurred.

`rescode.trm_max_iterations`

The optimizer terminated at the maximum number of iterations.

`rescode.trm_max_time`

The optimizer terminated at the maximum amount of time.

`rescode.trm_objective_range`

The optimizer terminated with an objective value outside the objective range.

`rescode.trm_mio_near_rel_gap`

The mixed-integer optimizer terminated as the delayed near optimal relative gap tolerance was satisfied.

`rescode.trm_mio_near_abs_gap`

The mixed-integer optimizer terminated as the delayed near optimal absolute gap tolerance was satisfied.

`rescode.trm_mio_num_relaxs`

The mixed-integer optimizer terminated as the maximum number of relaxations was reached.

`rescode.trm_mio_num_branches`

The mixed-integer optimizer terminated as the maximum number of branches was reached.

`rescode.trm_num_max_num_int_solutions`

The mixed-integer optimizer terminated as the maximum number of feasible solutions was reached.

`rescode.trm_stall`

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`

The optimizer terminated due to the return of the user-defined callback function.

`rescode.trm_max_num_setbacks`

The optimizer terminated as the maximum number of set-backs was reached. This indicates serious numerical problems and a possibly badly formulated problem.

`rescode.trm_numerical_problem`

The optimizer terminated due to numerical problems.

`rescode.trm_internal`

The optimizer terminated due to some internal reason. Please contact **MOSEK** support.

`rescode.trm_internal_stop`

The optimizer terminated for internal reasons. Please contact **MOSEK** support.

18.8.2 Warnings

`rescode.wrn_open_param_file`

The parameter file could not be opened.

`rescode.wrn_large_bound`

A numerically large bound value is specified.

`rescode.wrn_large_lo_bound`

A numerically large lower bound value is specified.

`rescode.wrn_large_up_bound`

A numerically large upper bound value is specified.

`rescode.wrn_large_con_fx`

An equality constraint is fixed to a numerically large value. This can cause numerical problems.

`rescode.wrn_large_cj`

A numerically large value is specified for one c_j .

`rescode.wrn_large_aij`

A numerically large value is specified for an $a_{i,j}$ element in A . The parameter `dparam.data_tol_aij_large` controls when an $a_{i,j}$ is considered large.

`rescode.wrn_zero_aij`

One or more zero elements are specified in A .

`rescode.wrn_name_max_len`

A name is longer than the buffer that is supposed to hold it.

`rescode.wrn_spar_max_len`

A value for a string parameter is longer than the buffer that is supposed to hold it.

`rescode.wrn_mps_split_rhs_vector`

An RHS vector is split into several nonadjacent parts in an MPS file.

`rescode.wrn_mps_split_ran_vector`

A RANGE vector is split into several nonadjacent parts in an MPS file.

`rescode.wrn_mps_split_bou_vector`

A BOUNDS vector is split into several nonadjacent parts in an MPS file.

`rescode.wrn_lp_old_quad_format`

Missing $/2$ after quadratic expressions in bound or objective.

`rescode.wrn_lp_drop_variable`

Ignored a variable because the variable was not previously defined. Usually this implies that a variable appears in the bound section but not in the objective or the constraints.

`rescode.wrn_nz_in_upr_tri`

Non-zero elements specified in the upper triangle of a matrix were ignored.

`rescode.wrn_dropped_nz_qobj`

One or more non-zero elements were dropped in the Q matrix in the objective.

`rescode.wrn_ignore_integer`

Ignored integer constraints.

`rescode.wrn_no_global_optimizer`

No global optimizer is available.

`rescode.wrn_mio_infeasible_final`

The final mixed-integer problem with all the integer variables fixed at their optimal values is infeasible.

`rescode.wrn_sol_filter`

Invalid solution filter is specified.

`rescode.wrn_undef_sol_file_name`
Undefined name occurred in a solution.

`rescode.wrn_sol_file_ignored_con`
One or more lines in the constraint section were ignored when reading a solution file.

`rescode.wrn_sol_file_ignored_var`
One or more lines in the variable section were ignored when reading a solution file.

`rescode.wrn_too_few_basis_vars`
An incomplete basis has been specified. Too few basis variables are specified.

`rescode.wrn_too_many_basis_vars`
A basis with too many variables has been specified.

`rescode.wrn_no_nonlinear_function_write`
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_license_expire`
The license expires.

`rescode.wrn_license_server`
The license server is not responding.

`rescode.wrn_empty_name`
A variable or constraint name is empty. The output file may be invalid.

`rescode.wrn_using_generic_names`
Generic names are used because a name is not valid. For instance when writing an LP file the names must not contain blanks or start with a digit.

`rescode.wrn_license_feature_expire`
The license expires.

`rescode.wrn_param_name_dou`
The parameter name is not recognized as a double parameter.

`rescode.wrn_param_name_int`
The parameter name is not recognized as a integer parameter.

`rescode.wrn_param_name_str`
The parameter name is not recognized as a string parameter.

`rescode.wrn_param_str_value`
The string is not recognized as a symbolic value for the parameter.

`rescode.wrn_param_ignored_cmio`
A parameter was ignored by the conic mixed integer optimizer.

`rescode.wrn_zeros_in_sparse_row`
One or more (near) zero elements are specified in a sparse row of a matrix. Since, it is redundant to specify zero elements then it may indicate an error.

`rescode.wrn_zeros_in_sparse_col`
One or more (near) zero elements are specified in a sparse column of a matrix. It is redundant to specify zero elements. Hence, it may indicate an error.

`rescode.wrn_incomplete_linear_dependency_check`
The linear dependency check(s) is incomplete. Normally this is not an important warning unless the optimization problem has been formulated with linear dependencies. Linear dependencies may prevent **MOSEK** from solving the problem.

`rescode.wrn_eliminator_space`
The eliminator is skipped at least once due to lack of space.

`rescode.wrn_presolve_outofspace`

The presolve is incomplete due to lack of space.

`rescode.wrn_write_changed_names`

Some names were changed because they were invalid for the output file format.

`rescode.wrn_write_discarded_cfix`

The fixed objective term could not be converted to a variable and was discarded in the output file.

`rescode.wrn_construct_solution_infeas`

After fixing the integer variables at the suggested values then the problem is infeasible.

`rescode.wrn_construct_invalid_sol_itg`

The initial value for one or more of the integer variables is not feasible.

`rescode.wrn_construct_no_sol_itg`

The construct solution requires an integer solution.

`rescode.wrn_duplicate_constraint_names`

Two constraint names are identical.

`rescode.wrn_duplicate_variable_names`

Two variable names are identical.

`rescode.wrn_duplicate_barvariable_names`

Two barvariable names are identical.

`rescode.wrn_duplicate_cone_names`

Two cone names are identical.

`rescode.wrn_ana_large_bounds`

This warning is issued by the problem analyzer, if one or more constraint or variable bounds are very large. One should consider omitting these bounds entirely by setting them to $+\text{inf}$ or $-\text{inf}$.

`rescode.wrn_ana_c_zero`

This warning is issued by the problem analyzer, if the coefficients in the linear part of the objective are all zero.

`rescode.wrn_ana_empty_cols`

This warning is issued by the problem analyzer, if columns, in which all coefficients are zero, are found.

`rescode.wrn_ana_close_bounds`

This warning is issued by problem analyzer, if ranged constraints or variables with very close upper and lower bounds are detected. One should consider treating such constraints as equalities and such variables as constants.

`rescode.wrn_ana_almost_int_bounds`

This warning is issued by the problem analyzer if a constraint is bound nearly integral.

`rescode.wrn_quad_cones_with_root_fixed_at_zero`

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`

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_no_dualizer`

No automatic dualizer is available for the specified problem. The primal problem is solved.

`rescode.wrn_sym_mat_large`

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.

18.8.3 Errors

`rescode.err_license`

Invalid license.

`rescode.err_license_expired`

The license has expired.

`rescode.err_license_version`

The license is valid for another version of **MOSEK**.

`rescode.err_size_license`

The problem is bigger than the license.

`rescode.err_prob_license`

The software is not licensed to solve the problem.

`rescode.err_file_license`

Invalid license file.

`rescode.err_missing_license_file`

MOSEK cannot license file or a token server. See the **MOSEK** installation manual for details.

`rescode.err_size_license_con`

The problem has too many constraints to be solved with the available license.

`rescode.err_size_license_var`

The problem has too many variables to be solved with the available license.

`rescode.err_size_license_intvar`

The problem contains too many integer variables to be solved with the available license.

`rescode.err_optimizer_license`

The optimizer required is not licensed.

`rescode.err_flexlm`

The FLEXlm license manager reported an error.

`rescode.err_license_server`

The license server is not responding.

`rescode.err_license_max`

Maximum number of licenses is reached.

`rescode.err_license_moseklm_daemon`

The MOSEKLM license manager daemon is not up and running.

`rescode.err_license_feature`

A requested feature is not available in the license file(s). Most likely due to an incorrect license system setup.

`rescode.err_platform_not_licensed`

A requested license feature is not available for the required platform.

`rescode.err_license_cannot_allocate`

The license system cannot allocate the memory required.

`rescode.err_license_cannot_connect`

MOSEK cannot connect to the license server. Most likely the license server is not up and running.

`rescode.err_license_invalid_hostid`

The host ID specified in the license file does not match the host ID of the computer.

`rescode.err_license_server_version`

The version specified in the checkout request is greater than the highest version number the daemon supports.

`rescode.err_license_no_server_support`

The license server does not support the requested feature. Possible reasons for this error include:

- The feature has expired.
- The feature's start date is later than today's date.
- The version requested is higher than feature's the highest supported version.
- A corrupted license file.

Try restarting the license and inspect the license server debug file, usually called `lmgrd.log`.

`rescode.err_license_no_server_line`

There is no `SERVER` line in the license file. All non-zero license count features need at least one `SERVER` line.

`rescode.err_open_dl`

A dynamic link library could not be opened.

`rescode.err_older_dll`

The dynamic link library is older than the specified version.

`rescode.err_newer_dll`

The dynamic link library is newer than the specified version.

`rescode.err_link_file_dll`

A file cannot be linked to a stream in the DLL version.

`rescode.err_thread_mutex_init`

Could not initialize a mutex.

`rescode.err_thread_mutex_lock`

Could not lock a mutex.

`rescode.err_thread_mutex_unlock`

Could not unlock a mutex.

`rescode.err_thread_create`

Could not create a thread. This error may occur if a large number of environments are created and not deleted again. In any case it is a good practice to minimize the number of environments created.

`rescode.err_thread_cond_init`

Could not initialize a condition.

`rescode.err_unknown`

Unknown error.

`rescode.err_space`

Out of space.

`rescode.err_file_open`

Error while opening a file.

`rescode.err_file_read`

File read error.

`rescode.err_file_write`

File write error.

`rescode.err_data_file_ext`

The data file format cannot be determined from the file name.

`rescode.err_invalid_file_name`

An invalid file name has been specified.

`rescode.err_invalid_sol_file_name`

An invalid file name has been specified.

`rescode.err_end_of_file`

End of file reached.

`rescode.err_null_env`
env is a NULL pointer.

`rescode.err_null_task`
task is a NULL pointer.

`rescode.err_invalid_stream`
An invalid stream is referenced.

`rescode.err_no_init_env`
env is not initialized.

`rescode.err_invalid_task`
The task is invalid.

`rescode.err_null_pointer`
An argument to a function is unexpectedly a NULL pointer.

`rescode.err_living_tasks`
All tasks associated with an environment must be deleted before the environment is deleted. There are still some undeleted tasks.

`rescode.err_blank_name`
An all blank name has been specified.

`rescode.err_dup_name`
The same name was used multiple times for the same problem item type.

`rescode.err_invalid_obj_name`
An invalid objective name is specified.

`rescode.err_invalid_con_name`
An invalid constraint name is used.

`rescode.err_invalid_var_name`
An invalid variable name is used.

`rescode.err_invalid_cone_name`
An invalid cone name is used.

`rescode.err_invalid_barvar_name`
An invalid symmetric matrix variable name is used.

`rescode.err_space_leaking`
MOSEK is leaking memory. This can be due to either an incorrect use of **MOSEK** or a bug.

`rescode.err_space_no_info`
No available information about the space usage.

`rescode.err_read_format`
The specified format cannot be read.

`rescode.err_mps_file`
An error occurred while reading an MPS file.

`rescode.err_mps_inv_field`
A field in the MPS file is invalid. Probably it is too wide.

`rescode.err_mps_inv_marker`
An invalid marker has been specified in the MPS file.

`rescode.err_mps_null_con_name`
An empty constraint name is used in an MPS file.

`rescode.err_mps_null_var_name`
An empty variable name is used in an MPS file.

`rescode.err_mps_undef_con_name`
An undefined constraint name occurred in an MPS file.

`rescode.err_mps_undef_var_name`
An undefined variable name occurred in an MPS file.

`rescode.err_mps_inv_con_key`
An invalid constraint key occurred in an MPS file.

`rescode.err_mps_inv_bound_key`
An invalid bound key occurred in an MPS file.

`rescode.err_mps_inv_sec_name`
An invalid section name occurred in an MPS file.

`rescode.err_mps_no_objective`
No objective is defined in an MPS file.

`rescode.err_mps splitted_var`
All elements in a column of the A matrix must be specified consecutively. Hence, it is illegal to specify non-zero elements in A for variable 1, then for variable 2 and then variable 1 again.

`rescode.err_mps_mul_con_name`
A constraint name was specified multiple times in the ROWS section.

`rescode.err_mps_mul_qsec`
Multiple QSECTIONs are specified for a constraint in the MPS data file.

`rescode.err_mps_mul_qobj`
The Q term in the objective is specified multiple times in the MPS data file.

`rescode.err_mps_inv_sec_order`
The sections in the MPS data file are not in the correct order.

`rescode.err_mps_mul_csec`
Multiple CSECTIONs are given the same name.

`rescode.err_mps_cone_type`
Invalid cone type specified in a CSECTION.

`rescode.err_mps_cone_overlap`
A variable is specified to be a member of several cones.

`rescode.err_mps_cone_repeat`
A variable is repeated within the CSECTION.

`rescode.err_mps_non_symmetric_q`
A non symmetric matrix has been specified.

`rescode.err_mps_duplicate_q_element`
Duplicate elements is specified in a Q matrix.

`rescode.err_mps_invalid_objsense`
An invalid objective sense is specified.

`rescode.err_mps_tab_in_field2`
A tab char occurred in field 2.

`rescode.err_mps_tab_in_field3`
A tab char occurred in field 3.

`rescode.err_mps_tab_in_field5`
A tab char occurred in field 5.

`rescode.err_mps_invalid_obj_name`
An invalid objective name is specified.

`rescode.err_lp_incompatible`
The problem cannot be written to an LP formatted file.

`rescode.err_lp_empty`
The problem cannot be written to an LP formatted file.

`rescode.err_lp_dup_slack_name`
The name of the slack variable added to a ranged constraint already exists.

`rescode.err_write_mps_invalid_name`
An invalid name is created while writing an MPS file. Usually this will make the MPS file unreadable.

`rescode.err_lp_invalid_var_name`
A variable name is invalid when used in an LP formatted file.

`rescode.err_lp_free_constraint`
Free constraints cannot be written in LP file format.

`rescode.err_write_opf_invalid_var_name`
Empty variable names cannot be written to OPF files.

`rescode.err_lp_file_format`
Syntax error in an LP file.

`rescode.err_write_lp_format`
Problem cannot be written as an LP file.

`rescode.err_read_lp_missing_end_tag`
Syntax error in LP file. Possibly missing End tag.

`rescode.err_lp_format`
Syntax error in an LP file.

`rescode.err_write_lp_non_unique_name`
An auto-generated name is not unique.

`rescode.err_read_lp_nonexisting_name`
A variable never occurred in objective or constraints.

`rescode.err_lp_write_conic_problem`
The problem contains cones that cannot be written to an LP formatted file.

`rescode.err_lp_write_geco_problem`
The problem contains general convex terms that cannot be written to an LP formatted file.

`rescode.err_writing_file`
An error occurred while writing file

`rescode.err_opf_format`
Syntax error in an OPF file

`rescode.err_opf_new_variable`
Introducing new variables is now allowed. When a `[variables]` section is present, it is not allowed to introduce new variables later in the problem.

`rescode.err_invalid_name_in_sol_file`
An invalid name occurred in a solution file.

`rescode.err_lp_invalid_con_name`
A constraint name is invalid when used in an LP formatted file.

`rescode.err_opf_premature_eof`
Premature end of file in an OPF file.

`rescode.err_json_syntax`
Syntax error in an JSON data

`rescode.err_json_string`
Error in JSON string.

`rescode.err_json_number_overflow`
Invalid number entry - wrong type or value overflow.

`rescode.err_json_format`
Error in an JSON Task file

`rescode.err_json_data`
Inconsistent data in JSON Task file

`rescode.err_json_missing_data`
Missing data section in JSON task file.

`rescode.err_argument_lenneq`
Incorrect length of arguments.

`rescode.err_argument_type`
Incorrect argument type.

`rescode.err_nr_arguments`
Incorrect number of function arguments.

`rescode.err_in_argument`
A function argument is incorrect.

`rescode.err_argument_dimension`
A function argument is of incorrect dimension.

`rescode.err_index_is_too_small`
An index in an argument is too small.

`rescode.err_index_is_too_large`
An index in an argument is too large.

`rescode.err_param_name`
The parameter name is not correct.

`rescode.err_param_name_dou`
The parameter name is not correct for a double parameter.

`rescode.err_param_name_int`
The parameter name is not correct for an integer parameter.

`rescode.err_param_name_str`
The parameter name is not correct for a string parameter.

`rescode.err_param_index`
Parameter index is out of range.

`rescode.err_param_is_too_large`
The parameter value is too large.

`rescode.err_param_is_too_small`
The parameter value is too small.

`rescode.err_param_value_str`
The parameter value string is incorrect.

`rescode.err_param_type`
The parameter type is invalid.

`rescode.err_inf_dou_index`
A double information index is out of range for the specified type.

`rescode.err_inf_int_index`
An integer information index is out of range for the specified type.

`rescode.err_index_arr_is_too_small`
An index in an array argument is too small.

`rescode.err_index_arr_is_too_large`
An index in an array argument is too large.

`rescode.err_inf_lint_index`
A long integer information index is out of range for the specified type.

`rescode.err_arg_is_too_small`
The value of a argument is too small.

`rescode.err_arg_is_too_large`
The value of a argument is too small.

`rescode.err_invalid_whichsol`
`whichsol` is invalid.

`rescode.err_inf_dou_name`
A double information name is invalid.

`rescode.err_inf_int_name`
An integer information name is invalid.

`rescode.err_inf_type`
The information type is invalid.

`rescode.err_inf_lint_name`
A long integer information name is invalid.

`rescode.err_index`
An index is out of range.

`rescode.err_whichsol`
The solution defined by `whichsol` does not exists.

`rescode.err_solitem`
The solution item number `solitem` is invalid. Please note that `solitem.snz` is invalid for the basic solution.

`rescode.err_whichitem_not_allowed`
`whichitem` is unacceptable.

`rescode.err_maxnumcon`
The maximum number of constraints specified is smaller than the number of constraints in the task.

`rescode.err_maxnumvar`
The maximum number of variables specified is smaller than the number of variables in the task.

`rescode.err_maxnumbarvar`
The maximum number of semidefinite variables specified is smaller than the number of semidefinite variables in the task.

`rescode.err_maxnumqnz`
The maximum number of non-zeros specified for the Q matrices is smaller than the number of non-zeros in the current Q matrices.

`rescode.err_too_small_max_num_nz`
The maximum number of non-zeros specified is too small.

`rescode.err_invalid_idx`
A specified index is invalid.

`rescode.err_invalid_max_num`
A specified index is invalid.

`rescode.err_numconlim`
Maximum number of constraints limit is exceeded.

`rescode.err_numvarlim`
Maximum number of variables limit is exceeded.

`rescode.err_too_small_maxnumanz`

The maximum number of non-zeros specified for A is smaller than the number of non-zeros in the current A .

`rescode.err_inv_aptre`

`aptre[j]` is strictly smaller than `aptrb[j]` for some j .

`rescode.err_mul_a_element`

An element in A is defined multiple times.

`rescode.err_inv_bk`

Invalid bound key.

`rescode.err_inv_bkc`

Invalid bound key is specified for a constraint.

`rescode.err_inv_bkx`

An invalid bound key is specified for a variable.

`rescode.err_inv_var_type`

An invalid variable type is specified for a variable.

`rescode.err_solver_proptype`

Problem type does not match the chosen optimizer.

`rescode.err_objective_range`

Empty objective range.

`rescode.err_first`

Invalid `first`.

`rescode.err_last`

Invalid index `last`. A given index was out of expected range.

`rescode.err_negative_surplus`

Negative surplus.

`rescode.err_negative_append`

Cannot append a negative number.

`rescode.err_undef_solution`

MOSEK has the following solution types:

- an interior-point solution,
- an basic solution,
- and an integer solution.

Each optimizer may set one or more of these solutions; e.g by default a successful optimization with the interior-point optimizer defines the interior-point solution, and, for linear problems, also the basic solution. This error occurs when asking for a solution or for information about a solution that is not defined.

`rescode.err_basis`

An invalid basis is specified. Either too many or too few basis variables are specified.

`rescode.err_inv_skc`

Invalid value in `skc`.

`rescode.err_inv_skx`

Invalid value in `skx`.

`rescode.err_inv_skn`

Invalid value in `skn`.

`rescode.err_inv_sk_str`

Invalid status key string encountered.

`rescode.err_inv_sk`
Invalid status key code.

`rescode.err_inv_cone_type_str`
Invalid cone type string encountered.

`rescode.err_inv_cone_type`
Invalid cone type code is encountered.

`rescode.err_invalid_surplus`
Invalid surplus.

`rescode.err_inv_name_item`
An invalid name item code is used.

`rescode.err_pro_item`
An invalid problem is used.

`rescode.err_invalid_format_type`
Invalid format type.

`rescode.err_firsti`
Invalid firsti.

`rescode.err_lasti`
Invalid lasti.

`rescode.err_firstj`
Invalid firstj.

`rescode.err_lastj`
Invalid lastj.

`rescode.err_max_len_is_too_small`
An maximum length that is too small has been specified.

`rescode.err_nonlinear_equality`
The model contains a nonlinear equality which defines a nonconvex set.

`rescode.err_nonconvex`
The optimization problem is nonconvex.

`rescode.err_nonlinear_ranged`
Nonlinear constraints with finite lower and upper bound always define a nonconvex feasible set.

`rescode.err_con_q_not_psd`
The quadratic constraint matrix is not positive semidefinite as expected for a constraint with finite upper bound. This results in a nonconvex problem. The parameter `dparam.check_convexity_rel_tol` can be used to relax the convexity check.

`rescode.err_con_q_not_nsd`
The quadratic constraint matrix is not negative semidefinite as expected for a constraint with finite lower bound. This results in a nonconvex problem. The parameter `dparam.check_convexity_rel_tol` can be used to relax the convexity check.

`rescode.err_obj_q_not_psd`
The quadratic coefficient matrix in the objective is not positive semidefinite as expected for a minimization problem. The parameter `dparam.check_convexity_rel_tol` can be used to relax the convexity check.

`rescode.err_obj_q_not_nsd`
The quadratic coefficient matrix in the objective is not negative semidefinite as expected for a maximization problem. The parameter `dparam.check_convexity_rel_tol` can be used to relax the convexity check.

`rescode.err_argument_perm_array`
An invalid permutation array is specified.

`rescode.err_cone_index`

An index of a non-existing cone has been specified.

`rescode.err_cone_size`

A cone with too few members is specified.

`rescode.err_cone_overlap`

One or more of the variables in the cone to be added is already member of another cone. Now assume the variable is x_j then add a new variable say x_k and the constraint

$$x_j = x_k$$

and then let x_k be member of the cone to be appended.

`rescode.err_cone_rep_var`

A variable is included multiple times in the cone.

`rescode.err_maxnumcone`

The value specified for `maxnumcone` is too small.

`rescode.err_cone_type`

Invalid cone type specified.

`rescode.err_cone_type_str`

Invalid cone type specified.

`rescode.err_cone_overlap_append`

The cone to be appended has one variable which is already member of another cone.

`rescode.err_remove_cone_variable`

A variable cannot be removed because it will make a cone invalid.

`rescode.err_sol_file_invalid_number`

An invalid number is specified in a solution file.

`rescode.err_huge_c`

A huge value in absolute size is specified for one c_j .

`rescode.err_huge_a_ij`

A numerically huge value is specified for an $a_{i,j}$ element in A . The parameter `dparam.data_tol_a_ij_huge` controls when an $a_{i,j}$ is considered huge.

`rescode.err_duplicate_a_ij`

An element in the A matrix is specified twice.

`rescode.err_lower_bound_is_a_nan`

The lower bound specified is not a number (nan).

`rescode.err_upper_bound_is_a_nan`

The upper bound specified is not a number (nan).

`rescode.err_infinite_bound`

A numerically huge bound value is specified.

`rescode.err_inv_qobj_subi`

Invalid value in `qosubi`.

`rescode.err_inv_qobj_subj`

Invalid value in `qosubj`.

`rescode.err_inv_qobj_val`

Invalid value in `qoval`.

`rescode.err_inv_qcon_subk`

Invalid value in `qcsubk`.

`rescode.err_inv_qcon_subi`

Invalid value in `qcsubi`.

`rescode.err_inv_qcon_subj`
Invalid value in `qconsubj`.

`rescode.err_inv_qcon_val`
Invalid value in `qconval`.

`rescode.err_qcon_subi_too_small`
Invalid value in `qconsubi`.

`rescode.err_qcon_subi_too_large`
Invalid value in `qconsubi`.

`rescode.err_qobj_upper_triangle`
An element in the upper triangle of Q^o is specified. Only elements in the lower triangle should be specified.

`rescode.err_qcon_upper_triangle`
An element in the upper triangle of a Q^k is specified. Only elements in the lower triangle should be specified.

`rescode.err_fixed_bound_values`
A fixed constraint/variable has been specified using the bound keys but the numerical value of the lower and upper bound is different.

`rescode.err_nonlinear_functions_not_allowed`
An operation that is invalid for problems with nonlinear functions defined has been attempted.

`rescode.err_user_func_ret`
An user function reported an error.

`rescode.err_user_func_ret_data`
An user function returned invalid data.

`rescode.err_user_nlo_func`
The user-defined nonlinear function reported an error.

`rescode.err_user_nlo_eval`
The user-defined nonlinear function reported an error.

`rescode.err_user_nlo_eval_hessubi`
The user-defined nonlinear function reported an invalid subscript in the Hessian.

`rescode.err_user_nlo_eval_hessubj`
The user-defined nonlinear function reported an invalid subscript in the Hessian.

`rescode.err_invalid_objective_sense`
An invalid objective sense is specified.

`rescode.err_undefined_objective_sense`
The objective sense has not been specified before the optimization.

`rescode.err_y_is_undefined`
The solution item y is undefined.

`rescode.err_nan_in_double_data`
An invalid floating point value was used in some double data.

`rescode.err_nan_in_blc`
 l^c contains an invalid floating point value, i.e. a NaN.

`rescode.err_nan_in_buc`
 u^c contains an invalid floating point value, i.e. a NaN.

`rescode.err_nan_in_c`
 c contains an invalid floating point value, i.e. a NaN.

`rescode.err_nan_in_blx`
 l^x contains an invalid floating point value, i.e. a NaN.

`rescode.err_nan_in_bux`

u^x contains an invalid floating point value, i.e. a NaN.

`rescode.err_invalid_aij`

$a_{i,j}$ contains an invalid floating point value, i.e. a NaN or an infinite value.

`rescode.err_sym_mat_invalid`

A symmetric matrix contains an invalid floating point value, i.e. a NaN or an infinite value.

`rescode.err_sym_mat_huge`

A symmetric matrix contains a huge value in absolute size. The parameter `dparam.data_sym_mat_tol_huge` controls when an $e_{i,j}$ is considered huge.

`rescode.err_inv_problem`

Invalid problem type. Probably a nonconvex problem has been specified.

`rescode.err_mixed_conic_and_nl`

The problem contains nonlinear terms conic constraints. The requested operation cannot be applied to this type of problem.

`rescode.err_global_inv_conic_problem`

The global optimizer can only be applied to problems without semidefinite variables.

`rescode.err_inv_optimizer`

An invalid optimizer has been chosen for the problem. This means that the simplex or the conic optimizer is chosen to optimize a nonlinear problem.

`rescode.err_mio_no_optimizer`

No optimizer is available for the current class of integer optimization problems.

`rescode.err_no_optimizer_var_type`

No optimizer is available for this class of optimization problems.

`rescode.err_final_solution`

An error occurred during the solution finalization.

`rescode.err_postsolve`

An error occurred during the postsolve. Please contact **MOSEK** support.

`rescode.err_overflow`

A computation produced an overflow i.e. a very large number.

`rescode.err_no_basis_sol`

No basic solution is defined.

`rescode.err_basis_factor`

The factorization of the basis is invalid.

`rescode.err_basis_singular`

The basis is singular and hence cannot be factored.

`rescode.err_factor`

An error occurred while factorizing a matrix.

`rescode.err_feasrepair_cannot_relax`

An optimization problem cannot be relaxed. This is the case e.g. for general nonlinear optimization problems.

`rescode.err_feasrepair_solving_relaxed`

The relaxed problem could not be solved to optimality. Please consult the log file for further details.

`rescode.err_feasrepair_inconsistent_bound`

The upper bound is less than the lower bound for a variable or a constraint. Please correct this before running the feasibility repair.

`rescode.err_repair_invalid_problem`

The feasibility repair does not support the specified problem type.

`rescode.err_repair_optimization_failed`

Computation the optimal relaxation failed. The cause may have been numerical problems.

`rescode.err_name_max_len`

A name is longer than the buffer that is supposed to hold it.

`rescode.err_name_is_null`

The name buffer is a NULL pointer.

`rescode.err_invalid_compression`

Invalid compression type.

`rescode.err_invalid_iomode`

Invalid io mode.

`rescode.err_no_primal_infeas_cer`

A certificate of primal infeasibility is not available.

`rescode.err_no_dual_infeas_cer`

A certificate of infeasibility is not available.

`rescode.err_no_solution_in_callback`

The required solution is not available.

`rescode.err_inv_marki`

Invalid value in marki.

`rescode.err_inv_markj`

Invalid value in markj.

`rescode.err_inv_numi`

Invalid numi.

`rescode.err_inv_numj`

Invalid numj.

`rescode.err_cannot_clone_nl`

A task with a nonlinear function callback cannot be cloned.

`rescode.err_cannot_handle_nl`

A function cannot handle a task with nonlinear function callbacks.

`rescode.err_invalid_accmode`

An invalid access mode is specified.

`rescode.err_task_incompatible`

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`

The Task file is invalid.

`rescode.err_task_write`

Failed to write the task file.

`rescode.err_lu_max_num_tries`

Could not compute the LU factors of the matrix within the maximum number of allowed tries.

`rescode.err_invalid_utf8`

An invalid UTF8 string is encountered.

`rescode.err_invalid_wchar`

An invalid wchar string is encountered.

`rescode.err_no_dual_for_itg_sol`

No dual information is available for the integer solution.

`rescode.err_no_snx_for_bas_sol`

s_n^x is not available for the basis solution.

`rescode.err_internal`
An internal error occurred. Please report this problem.

`rescode.err_api_array_too_small`
An input array was too short.

`rescode.err_api_cb_connect`
Failed to connect a callback object.

`rescode.err_api_fatal_error`
An internal error occurred in the API. Please report this problem.

`rescode.err_api_internal`
An internal fatal error occurred in an interface function.

`rescode.err_sen_format`
Syntax error in sensitivity analysis file.

`rescode.err_sen_undef_name`
An undefined name was encountered in the sensitivity analysis file.

`rescode.err_sen_index_range`
Index out of range in the sensitivity analysis file.

`rescode.err_sen_bound_invalid_up`
Analysis of upper bound requested for an index, where no upper bound exists.

`rescode.err_sen_bound_invalid_lo`
Analysis of lower bound requested for an index, where no lower bound exists.

`rescode.err_sen_index_invalid`
Invalid range given in the sensitivity file.

`rescode.err_sen_invalid_regexp`
Syntax error in regexp or regexp longer than 1024.

`rescode.err_sen_solution_status`
No optimal solution found to the original problem given for sensitivity analysis.

`rescode.err_sen_numerical`
Numerical difficulties encountered performing the sensitivity analysis.

`rescode.err_sen_unhandled_problem_type`
Sensitivity analysis cannot be performed for the specified problem. Sensitivity analysis is only possible for linear problems.

`rescode.err_unb_step_size`
A step size in an optimizer was unexpectedly unbounded. For instance, if the step-size becomes unbounded in phase 1 of the simplex algorithm then an error occurs. Normally this will happen only if the problem is badly formulated. Please contact **MOSEK** support if this error occurs.

`rescode.err_identical_tasks`
Some tasks related to this function call were identical. Unique tasks were expected.

`rescode.err_ad_invalid_codelist`
The code list data was invalid.

`rescode.err_internal_test_failed`
An internal unit test function failed.

`rescode.err_xml_invalid_problem_type`
The problem type is not supported by the XML format.

`rescode.err_invalid_ampl_stub`
Invalid AMPL stub.

`rescode.err_int64_to_int32_cast`
An 32 bit integer could not cast to a 64 bit integer.

`rescode.err_size_license_numcores`

The computer contains more cpu cores than the license allows for.

`rescode.err_infeas_undefined`

The requested value is not defined for this solution type.

`rescode.err_no_barx_for_solution`

There is no \bar{X} available for the solution specified. In particular note there are no \bar{X} defined for the basic and integer solutions.

`rescode.err_no_bars_for_solution`

There is no \bar{s} available for the solution specified. In particular note there are no \bar{s} defined for the basic and integer solutions.

`rescode.err_bar_var_dim`

The dimension of a symmetric matrix variable has to be greater than 0.

`rescode.err_sym_mat_invalid_row_index`

A row index specified for sparse symmetric matrix is invalid.

`rescode.err_sym_mat_invalid_col_index`

A column index specified for sparse symmetric matrix is invalid.

`rescode.err_sym_mat_not_lower_triangular`

Only the lower triangular part of sparse symmetric matrix should be specified.

`rescode.err_sym_mat_invalid_value`

The numerical value specified in a sparse symmetric matrix is not a floating value.

`rescode.err_sym_mat_duplicate`

A value in a symmetric matrix has been specified more than once.

`rescode.err_invalid_sym_mat_dim`

A sparse symmetric matrix of invalid dimension is specified.

`rescode.err_invalid_file_format_for_sym_mat`

The file format does not support a problem with symmetric matrix variables.

`rescode.err_invalid_file_format_for_cones`

The file format does not support a problem with conic constraints.

`rescode.err_invalid_file_format_for_general_nl`

The file format does not support a problem with general nonlinear terms.

`rescode.err_duplicate_constraint_names`

Two constraint names are identical.

`rescode.err_duplicate_variable_names`

Two variable names are identical.

`rescode.err_duplicate_barvariable_names`

Two barvariable names are identical.

`rescode.err_duplicate_cone_names`

Two cone names are identical.

`rescode.err_non_unique_array`

An array does not contain unique elements.

`rescode.err_argument_is_too_large`

The value of a function argument is too large.

`rescode.err_mio_internal`

A fatal error occurred in the mixed integer optimizer. Please contact **MOSEK** support.

`rescode.err_invalid_problem_type`

An invalid problem type.

`rescode.err_unhandled_solution_status`
Unhandled solution status.

`rescode.err_upper_triangle`
An element in the upper triangle of a lower triangular matrix is specified.

`rescode.err_lau_singular_matrix`
A matrix is singular.

`rescode.err_lau_not_positive_definite`
A matrix is not positive definite.

`rescode.err_lau_invalid_lower_triangular_matrix`
An invalid lower triangular matrix.

`rescode.err_lau_unknown`
An unknown error.

`rescode.err_lau_arg_m`
Invalid argument m.

`rescode.err_lau_arg_n`
Invalid argument n.

`rescode.err_lau_arg_k`
Invalid argument k.

`rescode.err_lau_arg_transa`
Invalid argument transa.

`rescode.err_lau_arg_transb`
Invalid argument transb.

`rescode.err_lau_arg_uplo`
Invalid argument uplo.

`rescode.err_lau_arg_trans`
Invalid argument trans.

`rescode.err_lau_invalid_sparse_symmetric_matrix`
An invalid sparse symmetric matrix is specified. Note only the lower triangular part with no duplicates is specified.

`rescode.err_cbf_parse`
An error occurred while parsing an CBF file.

`rescode.err_cbf_obj_sense`
An invalid objective sense is specified.

`rescode.err_cbf_no_variables`
No variables are specified.

`rescode.err_cbf_too_many_constraints`
Too many constraints specified.

`rescode.err_cbf_too_many_variables`
Too many variables specified.

`rescode.err_cbf_no_version_specified`
No version specified.

`rescode.err_cbf_syntax`
Invalid syntax.

`rescode.err_cbf_duplicate_obj`
Duplicate OBJ keyword.

`rescode.err_cbf_duplicate_con`
Duplicate CON keyword.

`rescode.err_cbf_duplicate_var`
Duplicate VAR keyword.

`rescode.err_cbf_duplicate_int`
Duplicate INT keyword.

`rescode.err_cbf_invalid_var_type`
Invalid variable type.

`rescode.err_cbf_invalid_con_type`
Invalid constraint type.

`rescode.err_cbf_invalid_domain_dimension`
Invalid domain dimension.

`rescode.err_cbf_duplicate_objacoord`
Duplicate index in OBJCOORD.

`rescode.err_cbf_duplicate_bcoord`
Duplicate index in BCOORD.

`rescode.err_cbf_duplicate_acoord`
Duplicate index in ACOORD.

`rescode.err_cbf_too_few_variables`
Too few variables defined.

`rescode.err_cbf_too_few_constraints`
Too few constraints defined.

`rescode.err_cbf_too_few_ints`
Too few ints are specified.

`rescode.err_cbf_too_many_ints`
Too many ints are specified.

`rescode.err_cbf_invalid_int_index`
Invalid INT index.

`rescode.err_cbf_unsupported`
Unsupported feature is present.

`rescode.err_cbf_duplicate_psdvar`
Duplicate PSDVAR keyword.

`rescode.err_cbf_invalid_psdvar_dimension`
Invalid PSDVAR dimension.

`rescode.err_cbf_too_few_psdvar`
Too few variables defined.

`rescode.err_mio_invalid_root_optimizer`
An invalid root optimizer was selected for the problem type.

`rescode.err_mio_invalid_node_optimizer`
An invalid node optimizer was selected for the problem type.

`rescode.err_toconic_constr_q_not_psd`
The matrix defining the quadratic part of constraint is not positive semidefinite.

`rescode.err_toconic_constraint_fx`
The quadratic constraint is an equality, thus not convex.

`rescode.err_toconic_constraint_ra`
The quadratic constraint has finite lower and upper bound, and therefore it is not convex.

`rescode.err_toconic_constr_not_conic`
The constraint is not conic representable.

`rescode.err_toconic_objective_not_psd`

The matrix defining the quadratic part of the objective function is not positive semidefinite.

`rescode.err_server_connect`

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`

Unexpected message or data from solver server.

`rescode.err_server_status`

Server returned non-ok HTTP status code

`rescode.err_server_token`

The job ID specified is incorrect or invalid

18.9 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 a finite upper bound.

`boundkey.fx`
The constraint or variable is fixed.

`boundkey.fr`
The constraint or variable is free.

`boundkey.ra`
The constraint or variable is ranged.

`mark`
Mark

`mark.lo`
The lower bound is selected for sensitivity analysis.

`mark.up`
The upper bound is selected for sensitivity analysis.

`simdegen`
Degeneracy strategies

`simdegen.none`
The simplex optimizer should use no degeneration strategy.

`simdegen.free`
The simplex optimizer chooses the degeneration strategy.

`simdegen.aggressive`
The simplex optimizer should use an aggressive degeneration strategy.

`simdegen.moderate`
The simplex optimizer should use a moderate degeneration strategy.

`simdegen.minimum`
The simplex optimizer should use a minimum degeneration strategy.

`transpose`
Transposed matrix.

`transpose.no`
No transpose is applied.

`transpose.yes`
A transpose is applied.

`uplo`
Triangular part of a symmetric matrix.

`uplo.lo`
Lower part.

`uplo.up`
Upper part

`simreform`
Problem reformulation.

`simreform.on`
Allow the simplex optimizer to reformulate the problem.

`simreform.off`
Disallow the simplex optimizer to reformulate the problem.

`simreform.free`
The simplex optimizer can choose freely.

`simreform.aggressive`
The simplex optimizer should use an aggressive reformulation strategy.

`simdupvec`
Exploit duplicate columns.

`simdupvec.on`
Allow the simplex optimizer to exploit duplicated columns.

`simdupvec.off`
Disallow the simplex optimizer to exploit duplicated columns.

`simdupvec.free`
The simplex optimizer can choose freely.

`simhotstart`
Hot-start type employed by the simplex optimizer

`simhotstart.none`
The simplex optimizer performs a coldstart.

`simhotstart.free`
The simplex optimizer chooses the hot-start type.

`simhotstart.status_keys`
Only the status keys of the constraints and variables are used to choose the type of hot-start.

`intpnthotstart`
Hot-start type employed by the interior-point optimizers.

`intpnthotstart.none`
The interior-point optimizer performs a coldstart.

`intpnthotstart.primal`
The interior-point optimizer exploits the primal solution only.

`intpnthotstart.dual`
The interior-point optimizer exploits the dual solution only.

`intpnthotstart.primal_dual`
The interior-point optimizer exploits both the primal and dual solution.

`callbackcode`
Progress callback codes

`callbackcode.begin_bi`
The basis identification procedure has been started.

`callbackcode.begin_conic`
The callback function is called when the conic optimizer is started.

`callbackcode.begin_dual_bi`
The callback function is called from within the basis identification procedure when the dual phase is started.

`callbackcode.begin_dual_sensitivity`
Dual sensitivity analysis is started.

`callbackcode.begin_dual_setup_bi`
The callback function is called when the dual BI phase is started.

`callbackcode.begin_dual_simplex`
The callback function is called when the dual simplex optimizer started.

`callbackcode.begin_dual_simplex_bi`
The callback function is called from within the basis identification procedure when the dual simplex clean-up phase is started.

`callbackcode.begin_full_convexity_check`
Begin full convexity check.

`callbackcode.begin_infeas_ana`

The callback function is called when the infeasibility analyzer is started.

`callbackcode.begin_intpnt`

The callback function is called when the interior-point optimizer is started.

`callbackcode.begin_license_wait`

Begin waiting for license.

`callbackcode.begin_mio`

The callback function is called when the mixed-integer optimizer is started.

`callbackcode.begin_optimizer`

The callback function is called when the optimizer is started.

`callbackcode.begin_presolve`

The callback function is called when the presolve is started.

`callbackcode.begin_primal_bi`

The callback function is called from within the basis identification procedure when the primal phase is started.

`callbackcode.begin_primal_repair`

Begin primal feasibility repair.

`callbackcode.begin_primal_sensitivity`

Primal sensitivity analysis is started.

`callbackcode.begin_primal_setup_bi`

The callback function is called when the primal BI setup is started.

`callbackcode.begin_primal_simplex`

The callback function is called when the primal simplex optimizer is started.

`callbackcode.begin_primal_simplex_bi`

The callback function is called from within the basis identification procedure when the primal simplex clean-up phase is started.

`callbackcode.begin_qcqp_reformulate`

Begin QCQP reformulation.

`callbackcode.begin_read`

MOSEK has started reading a problem file.

`callbackcode.begin_root_cutgen`

The callback function is called when root cut generation is started.

`callbackcode.begin_simplex`

The callback function is called when the simplex optimizer is started.

`callbackcode.begin_simplex_bi`

The callback function is called from within the basis identification procedure when the simplex clean-up phase is started.

`callbackcode.begin_to_conic`

Begin conic reformulation.

`callbackcode.begin_write`

MOSEK has started writing a problem file.

`callbackcode.conic`

The callback function is called from within the conic optimizer after the information database has been updated.

`callbackcode.dual_simplex`

The callback function is called from within the dual simplex optimizer.

`callbackcode.end_bi`

The callback function is called when the basis identification procedure is terminated.

`callbackcode.end_conic`

The callback function is called when the conic optimizer is terminated.

`callbackcode.end_dual_bi`

The callback function is called from within the basis identification procedure when the dual phase is terminated.

`callbackcode.end_dual_sensitivity`

Dual sensitivity analysis is terminated.

`callbackcode.end_dual_setup_bi`

The callback function is called when the dual BI phase is terminated.

`callbackcode.end_dual_simplex`

The callback function is called when the dual simplex optimizer is terminated.

`callbackcode.end_dual_simplex_bi`

The callback function is called from within the basis identification procedure when the dual clean-up phase is terminated.

`callbackcode.end_full_convexity_check`

End full convexity check.

`callbackcode.end_infeas_ana`

The callback function is called when the infeasibility analyzer is terminated.

`callbackcode.end_intpnt`

The callback function is called when the interior-point optimizer is terminated.

`callbackcode.end_license_wait`

End waiting for license.

`callbackcode.end_mio`

The callback function is called when the mixed-integer optimizer is terminated.

`callbackcode.end_optimizer`

The callback function is called when the optimizer is terminated.

`callbackcode.end_presolve`

The callback function is called when the presolve is completed.

`callbackcode.end_primal_bi`

The callback function is called from within the basis identification procedure when the primal phase is terminated.

`callbackcode.end_primal_repair`

End primal feasibility repair.

`callbackcode.end_primal_sensitivity`

Primal sensitivity analysis is terminated.

`callbackcode.end_primal_setup_bi`

The callback function is called when the primal BI setup is terminated.

`callbackcode.end_primal_simplex`

The callback function is called when the primal simplex optimizer is terminated.

`callbackcode.end_primal_simplex_bi`

The callback function is called from within the basis identification procedure when the primal clean-up phase is terminated.

`callbackcode.end_qcqp_reformulate`

End QCQP reformulation.

`callbackcode.end_read`

MOSEK has finished reading a problem file.

`callbackcode.end_root_cutgen`

The callback function is called when root cut generation is terminated.

`callbackcode.end_simplex`

The callback function is called when the simplex optimizer is terminated.

`callbackcode.end_simplex_bi`

The callback function is called from within the basis identification procedure when the simplex clean-up phase is terminated.

`callbackcode.end_to_conic`

End conic reformulation.

`callbackcode.end_write`

MOSEK has finished writing a problem file.

`callbackcode.im_bi`

The callback function is called from within the basis identification procedure at an intermediate point.

`callbackcode.im_conic`

The callback function is called at an intermediate stage within the conic optimizer where the information database has not been updated.

`callbackcode.im_dual_bi`

The callback function is called from within the basis identification procedure at an intermediate point in the dual phase.

`callbackcode.im_dual_sensitivity`

The callback function is called at an intermediate stage of the dual sensitivity analysis.

`callbackcode.im_dual_simplex`

The callback function is called at an intermediate point in the dual simplex optimizer.

`callbackcode.im_full_convexity_check`

The callback function is called at an intermediate stage of the full convexity check.

`callbackcode.im_intpnt`

The callback function is called at an intermediate stage within the interior-point optimizer where the information database has not been updated.

`callbackcode.im_license_wait`

MOSEK is waiting for a license.

`callbackcode.im_lu`

The callback function is called from within the LU factorization procedure at an intermediate point.

`callbackcode.im_mio`

The callback function is called at an intermediate point in the mixed-integer optimizer.

`callbackcode.im_mio_dual_simplex`

The callback function is called at an intermediate point in the mixed-integer optimizer while running the dual simplex optimizer.

`callbackcode.im_mio_intpnt`

The callback function is called at an intermediate point in the mixed-integer optimizer while running the interior-point optimizer.

`callbackcode.im_mio_primal_simplex`

The callback function is called at an intermediate point in the mixed-integer optimizer while running the primal simplex optimizer.

`callbackcode.im_order`

The callback function is called from within the matrix ordering procedure at an intermediate point.

`callbackcode.im_presolve`

The callback function is called from within the presolve procedure at an intermediate stage.

`callbackcode.im_primal_bi`

The callback function is called from within the basis identification procedure at an intermediate point in the primal phase.

`callbackcode.im_primal_sensitivity`

The callback function is called at an intermediate stage of the primal sensitivity analysis.

`callbackcode.im_primal_simplex`

The callback function is called at an intermediate point in the primal simplex optimizer.

`callbackcode.im_qo_reformulate`

The callback function is called at an intermediate stage of the conic quadratic reformulation.

`callbackcode.im_read`

Intermediate stage in reading.

`callbackcode.im_root_cutgen`

The callback is called from within root cut generation at an intermediate stage.

`callbackcode.im_simplex`

The callback function is called from within the simplex optimizer at an intermediate point.

`callbackcode.im_simplex_bi`

The callback function is called from within the basis identification procedure at an intermediate point in the simplex clean-up phase. The frequency of the callbacks is controlled by the *iparam.log_sim_freq* parameter.

`callbackcode.intpnt`

The callback function is called from within the interior-point optimizer after the information database has been updated.

`callbackcode.new_int_mio`

The callback function is called after a new integer solution has been located by the mixed-integer optimizer.

`callbackcode.primal_simplex`

The callback function is called from within the primal simplex optimizer.

`callbackcode.read_opf`

The callback function is called from the OPF reader.

`callbackcode.read_opf_section`

A chunk of Q non-zeros has been read from a problem file.

`callbackcode.solving_remote`

The callback function is called while the task is being solved on a remote server.

`callbackcode.update_dual_bi`

The callback function is called from within the basis identification procedure at an intermediate point in the dual phase.

`callbackcode.update_dual_simplex`

The callback function is called in the dual simplex optimizer.

`callbackcode.update_dual_simplex_bi`

The callback function is called from within the basis identification procedure at an intermediate point in the dual simplex clean-up phase. The frequency of the callbacks is controlled by the *iparam.log_sim_freq* parameter.

`callbackcode.update_presolve`

The callback function is called from within the presolve procedure.

`callbackcode.update_primal_bi`

The callback function is called from within the basis identification procedure at an intermediate point in the primal phase.

`callbackcode.update_primal_simplex`

The callback function is called in the primal simplex optimizer.

`callbackcode.update_primal_simplex_bi`

The callback function is called from within the basis identification procedure at an intermediate point in the primal simplex clean-up phase. The frequency of the callbacks is controlled by the *iparam.log_sim_freq* parameter.

`callbackcode.write_opf`

The callback function is called from the OPF writer.

`checkconvexitytype`

Types of convexity checks.

`checkconvexitytype.none`

No convexity check.

`checkconvexitytype.simple`

Perform simple and fast convexity check.

`checkconvexitytype.full`

Perform a full convexity check.

`compresstype`

Compression types

`compresstype.none`

No compression is used.

`compresstype.free`

The type of compression used is chosen automatically.

`compresstype.gzip`

The type of compression used is gzip compatible.

`conetype`

Cone types

`conetype.quad`

The cone is a quadratic cone.

`conetype.rquad`

The cone is a rotated quadratic cone.

`nametype`

Name types

`nametype.gen`

General names. However, no duplicate and blank names are allowed.

`nametype.mps`

MPS type names.

`nametype.lp`

LP type names.

`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_clean_dual_time`
Time spent within the dual clean-up optimizer of the basis identification procedure since its invocation.

`dinfitem.bi_clean_primal_time`
Time spent within the primal clean-up optimizer of the basis identification procedure since its invocation.

`dinfitem.bi_clean_time`
Time spent within the clean-up phase of the basis identification procedure since its invocation.

`dinfitem.bi_dual_time`
Time spent within the dual phase basis identification procedure since its invocation.

`dinfitem.bi_primal_time`
Time spent within the primal phase of the basis identification procedure since its invocation.

`dinfitem.bi_time`
Time spent within the basis identification procedure since its invocation.

`dinfitem.intpnt_dual_feas`
Dual feasibility measure reported by the interior-point optimizer. (For the interior-point optimizer this measure is not directly related to the original problem because a homogeneous model is employed.)

`dinfitem.intpnt_dual_obj`
Dual objective value reported by the interior-point optimizer.

`dinfitem.intpnt_factor_num_flops`
An estimate of the number of flops used in the factorization.

`dinfitem.intpnt_opt_status`
A measure of optimality of the solution. It should converge to +1 if the problem has a primal-dual optimal solution, and converge to -1 if the problem is (strictly) primal or dual infeasible. If the measure converges to another constant, or fails to settle, the problem is usually ill-posed.

`dinfitem.intpnt_order_time`
Order time (in seconds).

`dinfitem.intpnt_primal_feas`
Primal feasibility measure reported by the interior-point optimizer. (For the interior-point optimizer this measure is not directly related to the original problem because a homogeneous model is employed).

`dinfitem.intpnt_primal_obj`
Primal objective value reported by the interior-point optimizer.

`dinfitem.intpnt_time`
Time spent within the interior-point optimizer since its invocation.

`dinfitem.mio_clique_separation_time`
Seperation time for clique cuts.

`dinfitem.mio_cmir_separation_time`
Seperation time for CMIR cuts.

`dinfitem.mio_construct_solution_obj`
If **MOSEK** has successfully constructed an integer feasible solution, then this item contains the optimal objective value corresponding to the feasible solution.

`dinfitem.mio_dual_bound_after_presolve`
Value of the dual bound after presolve but before cut generation.

`dinfitem.mio_gmi_separation_time`
Seperation time for GMI cuts.

`dinfitem.mio_heuristic_time`
Total time spent in the optimizer.

`dinfitem.mio_implied_bound_time`
Seperation time for implied bound cuts.

`dinfitem.mio_knapsack_cover_separation_time`
Seperation time for knapsack cover.

`dinfitem.mio_obj_abs_gap`
Given the mixed-integer optimizer has computed a feasible solution and a bound on the optimal objective value, then this item contains the absolute gap defined by

$$|(\text{objective value of feasible solution}) - (\text{objective bound})|.$$

Otherwise it has the value -1.0.

`dinfitem.mio_obj_bound`
The best known bound on the objective function. This value is undefined until at least one relaxation has been solved: To see if this is the case check that `dinfitem.mio_num_relax` is strictly positive.

`dinfitem.mio_obj_int`
The primal objective value corresponding to the best integer feasible solution. Please note that at least one integer feasible solution must have been located i.e. check `dinfitem.mio_num_int_solutions`.

`dinfitem.mio_obj_rel_gap`
Given that the mixed-integer optimizer has computed a feasible solution and a bound on the optimal objective value, then this item contains the relative gap defined by

$$\frac{|(\text{objective value of feasible solution}) - (\text{objective bound})|}{\max(\delta, |(\text{objective value of feasible solution})|)}.$$

where δ is given by the parameter `dparam.mio_rel_gap_const`. Otherwise it has the value -1.0.

`dinfitem.mio_optimizer_time`
Total time spent in the optimizer.

`dinfitem.mio_probing_time`
Total time for probing.

`dinfitem.mio_root_cutgen_time`
Total time for cut generation.

`dinfitem.mio_root_optimizer_time`
Time spent in the optimizer while solving the root relaxation.

`dinfitem.mio_root_presolve_time`
Time spent in while presolving the root relaxation.

`dinfitem.mio_time`
Time spent in the mixed-integer optimizer.

`dinfitem.mio_user_obj_cut`
If the objective cut is used, then this information item has the value of the cut.

`dinfitem.optimizer_time`
Total time spent in the optimizer since it was invoked.

`dinfitem.presolve_eli_time`
Total time spent in the eliminator since the presolve was invoked.

`dinfitem.presolve_lindep_time`
Total time spent in the linear dependency checker since the presolve was invoked.

`dinfitem.presolve_time`
Total time (in seconds) spent in the presolve since it was invoked.

`dinfitem.primal_repair_penalty_obj`
The optimal objective value of the penalty function.

`dinfitem.qcqp_reformulate_max_perturbation`
Maximum absolute diagonal perturbation occurring during the QCQP reformulation.

`dinfitem.qcqp_reformulate_time`
Time spent with conic quadratic reformulation.

`dinfitem.qcqp_reformulate_worst_cholesky_column_scaling`
Worst Cholesky column scaling.

`dinfitem.qcqp_reformulate_worst_cholesky_diag_scaling`
Worst Cholesky diagonal scaling.

`dinfitem.rd_time`
Time spent reading the data file.

`dinfitem.sim_dual_time`
Time spent in the dual simplex optimizer since invoking it.

`dinfitem.sim_feas`
Feasibility measure reported by the simplex optimizer.

`dinfitem.sim_obj`
Objective value reported by the simplex optimizer.

`dinfitem.sim_primal_time`
Time spent in the primal simplex optimizer since invoking it.

`dinfitem.sim_time`
Time spent in the simplex optimizer since invoking it.

`dinfitem.sol_bas_dual_obj`
Dual objective value of the basic solution.

`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_barx`
Infinity norm of \bar{X} in the basic solution.

`dinfitem.sol_bas_nrm_slc`
Infinity norm of s_l^c in the basic solution.

`dinfitem.sol_bas_nrm_slx`
Infinity norm of s_l^x in the basic solution.

`dinfitem.sol_bas_nrm_suc`
Infinity norm of s_u^c in the basic solution.

`dinfitem.sol_bas_nrm_sux`
Infinity norm of s_u^X in the basic solution.

`dinfitem.sol_bas_nrm_xc`
Infinity norm of x^c in the basic solution.

`dinfitem.sol_bas_nrm_xx`
Infinity norm of x^x in the basic solution.

`dinfitem.sol_bas_nrm_y`
Infinity norm of y in the basic solution.

`dinfitem.sol_bas_primal_obj`
Primal objective value of the basic solution.

`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_itg_nrm_barx`
Infinity norm of \bar{X} in the integer solution.

`dinfitem.sol_itg_nrm_xc`
Infinity norm of x^c in the integer solution.

`dinfitem.sol_itg_nrm_xx`
Infinity norm of x^x in the integer solution.

`dinfitem.sol_itg_primal_obj`
Primal objective value of the integer solution.

`dinfitem.sol_itg_pviolbarvar`
Maximal primal bound violation for \bar{X} in the integer solution.

`dinfitem.sol_itg_pviolcon`
Maximal primal bound violation for x^c 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_pviolvar`
Maximal primal bound violation for x^x in the integer solution.

`dinfitem.sol_itr_dual_obj`
Dual objective value of the interior-point solution.

`dinfitem.sol_itr_dviolbarvar`
Maximal dual bound violation for \bar{X} in the interior-point solution.

`dinfitem.sol_itr_dviolcon`
Maximal dual bound violation for x^c in the interior-point solution.

`dinfitem.sol_itr_dviolcones`
Maximal dual violation for dual conic constraints in the interior-point solution.

`dinfitem.sol_itr_dviolvar`
Maximal dual bound violation for x^x in the interior-point solution.

`dinfitem.sol_itr_nrm_bars`
Infinity norm of \bar{S} in the interior-point solution.

`dinfitem.sol_itr_nrm_barx`
Infinity norm of \bar{X} in the interior-point solution.

`dinfitem.sol_itr_nrm_slc`
Infinity norm of s_l^c in the interior-point solution.

`dinfitem.sol_itr_nrm_slx`
Infinity norm of s_l^x in the interior-point solution.

`dinfitem.sol_itr_nrm_snx`
Infinity norm of s_n^x in the interior-point solution.

`dinfitem.sol_itr_nrm_suc`
Infinity norm of s_u^c in the interior-point solution.

`dinfitem.sol_itr_nrm_sux`
Infinity norm of s_u^X in the interior-point solution.

`dinfitem.sol_itr_nrm_xc`
Infinity norm of x^c in the interior-point solution.

`dinfitem.sol_itr_nrm_xx`
Infinity norm of x^x in the interior-point solution.

`dinfitem.sol_itr_nrm_y`
Infinity norm of y in the interior-point solution.

`dinfitem.sol_itr_primal_obj`
Primal objective value of the interior-point solution.

`dinfitem.sol_itr_pviolbarvar`
Maximal primal bound violation for \bar{X} in the interior-point solution.

`dinfitem.sol_itr_pviolcon`
Maximal primal bound violation for x^c in the interior-point solution.

`dinfitem.sol_itr_pviolcones`
Maximal primal violation for primal conic constraints in the interior-point solution.

`dinfitem.sol_itr_pviolvar`
Maximal primal bound violation for x^x in the interior-point solution.

`dinfitem.to_conic_time`
Time spent in the last to conic reformulation.

`feature`
License feature

`feature.pts`
Base system.

`feature.pton`
Nonlinear extension.

`liinfitem`
Long integer information items.

`liinfitem.bi_clean_dual_deg_iter`
Number of dual degenerate clean iterations performed in the basis identification.

`liinfitem.bi_clean_dual_iter`
Number of dual clean iterations performed in the basis identification.

`liinfitem.bi_clean_primal_deg_iter`
 Number of primal degenerate clean iterations performed in the basis identification.

`liinfitem.bi_clean_primal_iter`
 Number of primal clean iterations performed in the basis identification.

`liinfitem.bi_dual_iter`
 Number of dual pivots performed in the basis identification.

`liinfitem.bi_primal_iter`
 Number of primal pivots performed in the basis identification.

`liinfitem.intpnt_factor_num_nz`
 Number of non-zeros in factorization.

`liinfitem.mio_intpnt_iter`
 Number of interior-point iterations performed by the mixed-integer optimizer.

`liinfitem.mio_presolved_anz`
 Number of non-zero entries in the constraint matrix of presolved problem.

`liinfitem.mio_sim_maxiter_setbacks`
 Number of times the the simplex optimizer has hit the maximum iteration limit when re-optimizing.

`liinfitem.mio_simplex_iter`
 Number of simplex iterations performed by the mixed-integer optimizer.

`liinfitem.rd_numanz`
 Number of non-zeros in A that is read.

`liinfitem.rd_numqnz`
 Number of Q non-zeros.

`iinfitem`
 Integer information items.

`iinfitem.ana_pro_num_con`
 Number of constraints in the problem.
 This value is set by *Task.analyzeproblem*.

`iinfitem.ana_pro_num_con_eq`
 Number of equality constraints.
 This value is set by *Task.analyzeproblem*.

`iinfitem.ana_pro_num_con_fr`
 Number of unbounded constraints.
 This value is set by *Task.analyzeproblem*.

`iinfitem.ana_pro_num_con_lo`
 Number of constraints with a lower bound and an infinite upper bound.
 This value is set by *Task.analyzeproblem*.

`iinfitem.ana_pro_num_con_ra`
 Number of constraints with finite lower and upper bounds.
 This value is set by *Task.analyzeproblem*.

`iinfitem.ana_pro_num_con_up`
 Number of constraints with an upper bound and an infinite lower bound.
 This value is set by *Task.analyzeproblem*.

`iinfitem.ana_pro_num_var`
 Number of variables in the problem.
 This value is set by *Task.analyzeproblem*.

`iinfitem.ana_pro_num_var_bin`
Number of binary (0-1) variables.
This value is set by *Task.analyzeproblem*.

`iinfitem.ana_pro_num_var_cont`
Number of continuous variables.
This value is set by *Task.analyzeproblem*.

`iinfitem.ana_pro_num_var_eq`
Number of fixed variables.
This value is set by *Task.analyzeproblem*.

`iinfitem.ana_pro_num_var_fr`
Number of free variables.
This value is set by *Task.analyzeproblem*.

`iinfitem.ana_pro_num_var_int`
Number of general integer variables.
This value is set by *Task.analyzeproblem*.

`iinfitem.ana_pro_num_var_lo`
Number of variables with a lower bound and an infinite upper bound.
This value is set by *Task.analyzeproblem*.

`iinfitem.ana_pro_num_var_ra`
Number of variables with finite lower and upper bounds.
This value is set by *Task.analyzeproblem*.

`iinfitem.ana_pro_num_var_up`
Number of variables with an upper bound and an infinite lower bound. This value is set by
This value is set by *Task.analyzeproblem*.

`iinfitem.intpnt_factor_dim_dense`
Dimension of the dense sub system in factorization.

`iinfitem.intpnt_iter`
Number of interior-point iterations since invoking the interior-point optimizer.

`iinfitem.intpnt_num_threads`
Number of threads that the interior-point optimizer is using.

`iinfitem.intpnt_solve_dual`
Non-zero if the interior-point optimizer is solving the dual problem.

`iinfitem.mio_absgap_satisfied`
Non-zero if absolute gap is within tolerances.

`iinfitem.mio_clique_table_size`
Size of the clique table.

`iinfitem.mio_construct_num_roundings`
Number of values in the integer solution that is rounded to an integer value.

`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_initial_solution`
Is non-zero if an initial integer solution is specified.

`iinfitem.mio_near_absgap_satisfied`
Non-zero if absolute gap is within relaxed tolerances.

`iinfitem.mio_near_relgap_satisfied`
Non-zero if relative gap is within relaxed tolerances.

`iinfitem.mio_node_depth`
Depth of the last node solved.

`iinfitem.mio_num_active_nodes`
Number of active branch bound nodes.

`iinfitem.mio_num_branch`
Number of branches performed during the optimization.

`iinfitem.mio_num_clique_cuts`
Number of clique cuts.

`iinfitem.mio_num_cmir_cuts`
Number of Complemented Mixed Integer Rounding (CMIR) cuts.

`iinfitem.mio_num_gomory_cuts`
Number of Gomory cuts.

`iinfitem.mio_num_implied_bound_cuts`
Number of implied bound cuts.

`iinfitem.mio_num_int_solutions`
Number of integer feasible solutions that has been found.

`iinfitem.mio_num_knapsack_cover_cuts`
Number of clique cuts.

`iinfitem.mio_num_relax`
Number of relaxations solved during the optimization.

`iinfitem.mio_num_repeated_presolve`
Number of times presolve was repeated at root.

`iinfitem.mio_numcon`
Number of constraints in the problem solved by the mixed-integer optimizer.

`iinfitem.mio_numint`
Number of integer variables 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_obj_bound_defined`
Non-zero if a valid objective bound has been found, otherwise zero.

`iinfitem.mio_presolved_numbin`
Number of binary variables in the problem solved by the mixed-integer optimizer.

`iinfitem.mio_presolved_numcon`
Number of constraints in the presolved problem.

`iinfitem.mio_presolved_numcont`
Number of continuous variables in the problem solved by the mixed-integer optimizer.

`iinfitem.mio_presolved_numint`
Number of integer variables in the presolved problem.

`iinfitem.mio_presolved_numvar`
Number of variables in the presolved problem.

`iinfitem.mio_relgap_satisfied`
Non-zero if relative gap is within tolerances.

`iinfitem.mio_total_num_cuts`
Total number of cuts generated by the mixed-integer optimizer.

`iinfitem.mio_user_obj_cut`
If it is non-zero, then the objective cut is used.

`iinfitem.opt_numcon`
Number of constraints in the problem solved when the optimizer is called.

`iinfitem.opt_numvar`
Number of variables in the problem solved when the optimizer is called

`iinfitem.optimize_response`
The response code returned by optimize.

`iinfitem.rd_numbarvar`
Number of variables read.

`iinfitem.rd_numcon`
Number of constraints read.

`iinfitem.rd_numcone`
Number of conic constraints read.

`iinfitem.rd_numintvar`
Number of integer-constrained variables read.

`iinfitem.rd_numq`
Number of nonempty Q matrices read.

`iinfitem.rd_numvar`
Number of variables read.

`iinfitem.rd_prototype`
Problem type.

`iinfitem.sim_dual_deg_iter`
The number of dual degenerate iterations.

`iinfitem.sim_dual_hotstart`
If 1 then the dual simplex algorithm is solving from an advanced basis.

`iinfitem.sim_dual_hotstart_lu`
If 1 then a valid basis factorization of full rank was located and used by the dual simplex algorithm.

`iinfitem.sim_dual_inf_iter`
The number of iterations taken with dual infeasibility.

`iinfitem.sim_dual_iter`
Number of dual simplex iterations during the last optimization.

`iinfitem.sim_numcon`
Number of constraints in the problem solved by the simplex optimizer.

`iinfitem.sim_numvar`
Number of variables in the problem solved by the simplex optimizer.

`iinfitem.sim_primal_deg_iter`
The number of primal degenerate iterations.

`iinfitem.sim_primal_hotstart`
If 1 then the primal simplex algorithm is solving from an advanced basis.

`iinfitem.sim_primal_hotstart_lu`
If 1 then a valid basis factorization of full rank was located and used by the primal simplex algorithm.

`iinfitem.sim_primal_inf_iter`
The number of iterations taken with primal infeasibility.

`iinfitem.sim_primal_iter`
Number of primal simplex iterations during the last optimization.

`iinfitem.sim_solve_dual`
Is non-zero if dual problem is solved.

`iinfitem.sol_bas_prosta`
Problem status of the basic solution. Updated after each optimization.

`iinfitem.sol_bas_solsta`
Solution status of the basic solution. Updated after each optimization.

`iinfitem.sol_itg_prosta`
Problem status of the integer solution. Updated after each optimization.

`iinfitem.sol_itg_solsta`
Solution status of the integer solution. Updated after each optimization.

`iinfitem.sol_itr_prosta`
Problem status of the interior-point solution. Updated after each optimization.

`iinfitem.sol_itr_solsta`
Solution status of the interior-point solution. Updated after each optimization.

`iinfitem.sto_num_a_realloc`
Number of times the storage for storing A has been changed. A large value may indicate that memory fragmentation may occur.

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.

`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.

`mpsformat.cplex`

The CPLEX compatible version of the MPS format is employed.

`objsense`

Objective sense types

`objsense.minimize`

The problem should be minimized.

`objsense.maximize`

The problem should be maximized.

`onoffkey`

On/off

`onoffkey.on`

Switch the option on.

`onoffkey.off`

Switch the option off.

`optimizertype`

Optimizer types

`optimizertype.conic`

The optimizer for problems having conic constraints.

`optimizertype.dual_simplex`

The dual simplex optimizer is used.

`optimizertype.free`

The optimizer is chosen automatically.

`optimizertype.free_simplex`

One of the simplex optimizers is used.

`optimizertype.intpnt`

The interior-point optimizer is used.

`optimizertype.mixed_int`

The mixed-integer optimizer.

`optimizertype.primal_simplex`

The primal simplex optimizer is used.

`orderingtype`

Ordering strategies

`orderingtype.free`

The ordering method is chosen automatically.

`orderingtype.appminloc`

Approximate minimum local fill-in ordering is employed.

`orderingtype.experimental`

This option should not be used.

`orderingtype.try_graphpar`

Always try the graph partitioning based ordering.

`orderingtype.force_graphpar`

Always use the graph partitioning based ordering even if it is worse than the approximate minimum local fill ordering.

`orderingtype.none`

No ordering is used.

`presolvemode`

Presolve method.

`presolvemode.off`

The problem is not presolved before it is optimized.

`presolvemode.on`

The problem is presolved before it is optimized.

`presolvemode.free`

It is decided automatically whether to presolve before the problem is optimized.

`parametertype`

Parameter type

`parametertype.invalid_type`

Not a valid parameter.

`parametertype.dou_type`

Is a double parameter.

`parametertype.int_type`

Is an integer parameter.

`parametertype.str_type`

Is a string parameter.

`problemitem`

Problem data items

`problemitem.var`

Item is a variable.

`problemitem.con`

Item is a constraint.

`problemitem.cone`

Item is a cone.

`problemtype`

Problem types

`problemtype.lo`

The problem is a linear optimization problem.

`problemtype.qo`

The problem is a quadratic optimization problem.

`problemtype.qcqp`

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.

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.

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.

`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.

soltype

Solution types

soltype.bas

The basic solution.

soltype.itr

The interior solution.

soltype.itg

The integer solution.

solveform

Solve primal or dual form

solveform.free

The optimizer is free to solve either the primal or the dual problem.

solveform.primal

The optimizer should solve the primal problem.

solveform.dual

The optimizer should solve the dual problem.

stakey

Status keys

stakey.unk

The status for the constraint or variable is unknown.

stakey.bas

The constraint or variable is in the basis.

stakey.supbas

The constraint or variable is super basic.

stakey.low

The constraint or variable is at its lower bound.

stakey.upr

The constraint or variable is at its upper bound.

stakey.fix

The constraint or variable is fixed.

stakey.inf

The constraint or variable is infeasible in the bounds.

startpointtype

Starting point types

startpointtype.free

The starting point is chosen automatically.

startpointtype.guess

The optimizer guesses a starting point.

startpointtype.constant

The optimizer constructs a starting point by assigning a constant value to all primal and dual variables. This starting point is normally robust.

startpointtype.satisfy_bounds

The starting point is chosen to satisfy all the simple bounds on nonlinear variables. If this starting point is employed, then more care than usual should be employed when choosing the bounds on the nonlinear variables. In particular very tight bounds should be avoided.

streamtype

Stream types

`streamtype.log`

Log stream. Contains the aggregated contents of all other streams. This means that a message written to any other stream will also be written to this stream.

`streamtype.msg`

Message stream. Log information relating to performance and progress of the optimization is written to this stream.

`streamtype.err`

Error stream. Error messages are written to this stream.

`streamtype.wrn`

Warning stream. Warning messages are written to this stream.

`value`

Integer values

`value.max_str_len`

Maximum string length allowed in **MOSEK**.

`value.license_buffer_length`

The length of a license key buffer.

`variabletype`

Variable types

`variabletype.type_cont`

Is a continuous variable.

`variabletype.type_int`

Is an integer variable.

18.10 Class types

`mosek.ItgSolutionCallback`

A handler class for integer solution callbacks.

`ItgSolutionCallback.callback`

```
void callback (double[] xx)
```

The integer solution callback is a user-defined function which will be called by **MOSEK** when it improves the best mixed-integer solution.

The user *must not* call any **MOSEK** function directly or indirectly from the callback function.

Parameters `xx (double[])` – An array with the values of all variables in the currently best solution. (input)

`mosek.Progress`

A handler class for progress callbacks.

`Progress.progress`

```
int progress (mosek.callbackcode code)
```

The progress callback is a user-defined function which will be called by **MOSEK** occasionally during the optimization process. In particular, the callback function is called at the beginning of each iteration in the interior-point optimizer. For the simplex optimizers `iparam.log_sim_freq` controls how frequently the callback is called.

The user *must not* call any **MOSEK** function directly or indirectly from the callback function.

Parameters `code` (*callbackcode*) – Callback code indicating current operation of the solver. (input)

Return (`int`) – Non-zero if the optimizer should be stopped; zero otherwise.

`mosek.Stream`

A stream handler class.

`Stream.stream`

```
void stream (String msg)
```

The message-stream callback function is a user-defined function which can be linked to any of the **MOSEK** streams. Doing so, the function is called whenever **MOSEK** sends a message to the stream.

The user *must not* call any **MOSEK** function directly or indirectly from the callback function.

Parameters `msg` (`String`) – A message passed to the handler. (input)

`mosek.DataCallback`

A handler class for data callbacks.

`DataCallback.callback`

```
int callback
(mosek.callbackcode code,
 double[] dinf,
 int[] iinf,
 long[] liinf)
```

The data callback is a user-defined function which will be called by **MOSEK** occasionally during the optimization process. In particular, the callback function is called at the beginning of each iteration in the interior-point optimizer. For the simplex optimizers *iparam.log_sim_freq* controls how frequently the callback is called.

The user *must not* call any **MOSEK** function directly or indirectly from the callback function. The only exception is the possibility to retrieve an integer solution, see 7.1.

Parameters

- `code` (*callbackcode*) – Callback code indicating current operation of the solver. (input)
- `dinf` (`double[]`) – Array of double information items. (input)
- `iinf` (`int[]`) – Array of integer information items. (input)
- `liinf` (`long[]`) – Array of long integer information items. (input)

Return (`int`) – Non-zero if the optimizer should be stopped; zero otherwise.

18.11 Nonlinear extensions

18.11.1 Separable Convex Optimization (SCopt)

SCopt is an easy-to-use interface to the nonlinear optimizer when solving separable convex problems. See Section 6.1 for a tutorial and example code. As currently implemented, SCopt can handle only the nonlinear expressions $x \ln(x)$, e^x , $\ln(x)$, and x^g . However, it should be fairly easy to extend the interface to other nonlinear function of a single variable if needed.

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. Every nonlinear expression added to the objective should be specified by a 5-tuple of parameters:

<code>opro[k]</code>	<code>oprjo[k]</code>	<code>oprfo[k]</code>	<code>oprgo[k]</code>	<code>oprho[k]</code>	Expression added in objective
<code>scopt.ent</code>	j	f	g	h	$f x_j \ln(x_j)$
<code>scopt.exp</code>	j	f	g	h	$f e^{g x_j + h}$
<code>scopt.log</code>	j	f	g	h	$f \ln(g x_j + h)$
<code>scopt.pow</code>	j	f	g	h	$f (x_j + h)^g$

Every nonlinear expression added to the constraints should be specified by a 6-tuple of parameters:

<code>oprc[k]</code>	<code>opric[k]</code>	<code>oprjc[k]</code>	<code>oprfc[k]</code>	<code>oprgc[k]</code>	<code>oprhc[k]</code>	Expression added to constraint i
<code>scopt.ent</code>	i	j	f	g	h	$f x_j \ln(x_j)$
<code>scopt.exp</code>	i	j	f	g	h	$f e^{g x_j + h}$
<code>scopt.log</code>	i	j	f	g	h	$f \ln(g x_j + h)$
<code>scopt.pow</code>	i	j	f	g	h	$f (x_j + h)^g$

In each case `opr` specifies the kind of expression to be added, `oprf`, `oprg` and `oprh` are the parameters and `opri`, `oprj` determine the variable and/or constraint to be considered. The concrete API specification follows.

`scopr`

Type of nonlinear term in the SCopt interface.

`scopt.ent`

Entropy function $f x \ln(x)$

`scopt.exp`

Exponential function $f e^{g x + h}$

`scopt.log`

Logarithm $f \ln(g x + h)$

`scopt.pow`

Power function $f (x + h)^g$

`Task.putSCeval`

```
void putSCeval(
    scopr[]  opro,
    int[]    oprjo,
    double[] oprfo,
    double[] oprgo,
    double[] oprho,
    scopr[]  oprc,
    int[]    opric,
    int[]    oprjc,
    double[] oprfc,
    double[] oprgc,
    double[] oprhc)
```

Define the nonlinear part of the problem in the format specified by the SCopt interface. The first five arguments describe the nonlinear terms added to the objective, and should have the same length. The remaining six arguments describe the nonlinear terms added to the constraints and should have the same length. Multiple terms involving the same variable and constraint are possible, they will be added up.

Parameters

- `opro` (`scopr[]`) – List of function indicators defining the objective terms. (input)
- `oprjo` (`int[]`) – List of variable indexes for the objective terms. (input)

- `oprfo` (`double[]`) – List of f values for the objective terms. (input)
- `oprgo` (`double[]`) – List of g values for the objective terms. (input)
- `oprho` (`double[]`) – List of h values for the objective terms. (input)
- `oprc` (`scopr[]`) – List of function indicators defining the constraint terms. (input)
- `opric` (`int[]`) – List of constraint indexes for the constraint terms. (input)
- `oprjc` (`int[]`) – List of variable indexes for the constraint terms. (input)
- `oprfc` (`double[]`) – List of f values for the constraint terms. (input)
- `oprgc` (`double[]`) – List of g values for the constraint terms. (input)
- `oprhc` (`double[]`) – List of h values for the constraint terms. (input)

`Task.clearSCeval`

```
void clearSCeval ()
```

Remove all non-linear separable terms from the task.

`Task.writeSC`

```
void writeSC  
  (String scfilename,  
   String taskfilename)
```

Write problem to an SCopt file and a normal problem file.

Parameters

- `scfilename` (`String`) – Name of SCopt terms file. (input)
- `taskfilename` (`String`) – Name of problem file. (input)

SUPPORTED FILE FORMATS

MOSEK supports a range of problem and solution formats listed in [Table 19.1](#) and [Table 19.2](#). 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 19.1](#).

Table 19.1: List of supported file formats for optimization problems.

Format Type	Ext.	Binary/Text	LP	QO	CQO	SDP
<i>LP</i>	lp	plain text	X	X		
<i>MPS</i>	mps	plain text	X	X		
<i>OPF</i>	opf	plain text	X	X	X	
<i>CBF</i>	cbf	plain text	X		X	X
<i>OSiL</i>	xml	xml text	X	X		
<i>Task format</i>	task	binary	X	X	X	X
<i>Jtask format</i>	jtask	text	X	X	X	X

Solution formats

See [Table 19.2](#).

Table 19.2: List of supported solution formats.

Format Type	Ext.	Binary/Text	Description
<i>SOL</i>	sol	plain text	Interior Solution
	bas	plain text	Basic Solution
	int	plain text	Integer
<i>Jsol format</i>	jsol	text	Solution

Compression

MOSEK supports GZIP 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

problem.mps.gz

will be considered as a GZIP compressed MPS file.

19.1 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

$$\begin{array}{llll} \text{minimize/maximize} & & c^T x + \frac{1}{2} q^o(x) & \\ \text{subject to} & l^c \leq & Ax + \frac{1}{2} q(x) & \leq u^c, \\ & l^x \leq & x & \leq u^x, \\ & & x_{\mathcal{J}} & \text{integer,} \end{array}$$

where

- $x \in \mathbb{R}^n$ is the vector of decision variables.
- $c \in \mathbb{R}^n$ is the linear term in the objective.
- $q^o : \mathbb{R}^n \rightarrow \mathbb{R}$ is the quadratic term in the objective where

$$q^o(x) = x^T Q^o x$$

and it is assumed that

$$Q^o = (Q^o)^T.$$

- $A \in \mathbb{R}^{m \times n}$ is the constraint matrix.
- $l^c \in \mathbb{R}^m$ is the lower limit on the activity for the constraints.
- $u^c \in \mathbb{R}^m$ is the upper limit on the activity for the constraints.
- $l^x \in \mathbb{R}^n$ is the lower limit on the activity for the variables.
- $u^x \in \mathbb{R}^n$ is the upper limit on the activity for the variables.
- $q : \mathbb{R}^n \rightarrow \mathbb{R}$ is a vector of quadratic functions. Hence,

$$q_i(x) = x^T Q^i x$$

where it is assumed that

$$Q^i = (Q^i)^T.$$

- $\mathcal{J} \subseteq \{1, 2, \dots, n\}$ is an index set of the integer constrained variables.

19.1.1 File Sections

An LP formatted file contains a number of sections specifying the objective, constraints, variable bounds, and variable types. The section keywords may be any mix of upper and lower case letters.

Objective Function

The first section beginning with one of the keywords

```

max
maximum
maximize
min
minimum
minimize

```

defines the objective sense and the objective function, i.e.

$$c^T x + \frac{1}{2} x^T Q^o x.$$

The objective may be given a name by writing

```
myname:
```

before the expressions. If no name is given, then the objective is named `obj`.

The objective function contains linear and quadratic terms. The linear terms are written as:

```
4 x1 + x2 - 0.1 x3
```

and so forth. The quadratic terms are written in square brackets (`[]`) and are either squared or multiplied as in the examples

```
x1^2
```

and

```
x1 * x2
```

There may be zero or more pairs of brackets containing quadratic expressions.

An example of an objective section is

```

minimize
myobj: 4 x1 + x2 - 0.1 x3 + [ x1^2 + 2.1 x1 * x2 ]/2

```

Please note that the quadratic expressions are multiplied with $\frac{1}{2}$, so that the above expression means

$$\text{minimize } 4x_1 + x_2 - 0.1 \cdot x_3 + \frac{1}{2}(x_1^2 + 2.1 \cdot x_1 \cdot x_2)$$

If the same variable occurs more than once in the linear part, the coefficients are added, so that `4 x1 + 2 x1` is equivalent to `6 x1`. In the quadratic expressions `x1 * x2` is equivalent to `x2 * x1` and, as in the linear part, if the same variables multiplied or squared occur several times their coefficients are added.

Constraints

The second section beginning with one of the keywords

```

subj to
subject to
s.t.
st

```

defines the linear constraint matrix A and the quadratic matrices Q^i .

A constraint contains a name (optional), expressions adhering to the same rules as in the objective and a bound:

```

subject to
con1: x1 + x2 + [ x3^2 ]/2 <= 5.1

```

The bound type (here \leq) may be any of $<$, \leq , $=$, $>$, \geq ($<$ and \leq mean the same), and the bound may be any number.

In the standard LP format it is not possible to define more than one bound, but **MOSEK** supports defining ranged constraints by using double-colon ($::$) instead of a single-colon ($:$) after the constraint name, i.e.

$$-5 \leq x_1 + x_2 \leq 5 \quad (19.1)$$

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 (19.1) may be written as

$$x_1 + x_2 - sl_1 = 0, \quad -5 \leq sl_1 \leq 5.$$

Bounds

Bounds on the variables can be specified in the bound section beginning with one of the keywords

```
bound
bounds
```

The bounds section is optional but should, if present, follow the **subject to** section. All variables listed in the bounds section must occur in either the objective or a constraint.

The default lower and upper bounds are 0 and $+\infty$. A variable may be declared free with the keyword **free**, which means that the lower bound is $-\infty$ and the upper bound is $+\infty$. Furthermore it may be assigned a finite lower and upper bound. The bound definitions for a given variable may be written in one or two lines, and bounds can be any number or $\pm\infty$ (written as **+inf/-inf/+infinity/-infinity**) as in the example

```
bounds
x1 free
x2 <= 5
0.1 <= x2
x3 = 42
2 <= x4 < +inf
```

Variable Types

The final two sections are optional and must begin with one of the keywords

```
bin
binaries
binary
```

and

```
gen
general
```

Under **general** all integer variables are listed, and under **binary** all binary (integer variables with bounds 0 and 1) are listed:

```

general
x1 x2
binary
x3 x4

```

Again, all variables listed in the binary or general sections must occur in either the objective or a constraint.

Terminating Section

Finally, an LP formatted file must be terminated with the keyword

```
end
```

19.1.2 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

```

19.1.3 LP Format peculiarities

Comments

Anything on a line after a \ is ignored and is treated as a comment.

Names

A name for an objective, a constraint or a variable may contain the letters *a-z*, *A-Z*, the digits *0-9* and the characters

!"#\$%&()/,.;?@_`'|~

The first character in a name must not be a number, a period or the letter *e* or *E*. Keywords must not be used as names.

MOSEK accepts any character as valid for names, except \0. A name that is not allowed in LP file will be changed and a warning will be issued.

The algorithm for making names LP valid works as follows: The name is interpreted as an **utf-8** string. For a unicode character *c*:

- If *c*==_ (underscore), the output is __ (two underscores).
- If *c* is a valid LP name character, the output is just *c*.
- If *c* is another character in the ASCII range, the output is _XX, where XX is the hexadecimal code for the character.
- If *c* is a character in the range 127-65535, the output is _uXXXX, where XXXX is the hexadecimal code for the character.
- If *c* is a character above 65535, the output is _UXXXXXXXX, where XXXXXXXX is the hexadecimal code for the character.

Invalid **utf-8** substrings are escaped as _XX', and if a name starts with a period, *e* or *E*, that character is escaped as _XX.

Variable Bounds

Specifying several upper or lower bounds on one variable is possible but **MOSEK** uses only the tightest bounds. If a variable is fixed (with =), then it is considered the tightest bound.

MOSEK Extensions to the LP Format

Some optimization software packages employ a more strict definition of the LP format than the one used by **MOSEK**. The limitations imposed by the strict LP format are the following:

- Quadratic terms in the constraints are not allowed.
- Names can be only 16 characters long.
- Lines must not exceed 255 characters in length.

If an LP formatted file created by **MOSEK** should satisfy the strict definition, then the parameter

- *iparam.write_lp_strict_format*

should be set; note, however, that some problems cannot be written correctly as a strict LP formatted file. For instance, all names are truncated to 16 characters and hence they may lose their uniqueness and change the problem.

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*

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.

19.1.4 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.

19.1.5 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).

19.2 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].

19.2.1 MPS File Structure

The version of the MPS format supported by **MOSEK** allows specification of an optimization problem of the form

$$\begin{aligned} l^c &\leq Ax + q(x) &\leq u^c, \\ l^x &\leq x &\leq u^x, \\ &x \in \mathcal{K}, \\ &x_{\mathcal{I}} \text{ integer}, \end{aligned} \tag{19.2}$$

where

- $x \in \mathbb{R}^n$ is the vector of decision variables.
- $A \in \mathbb{R}^{m \times n}$ is the constraint matrix.
- $l^c \in \mathbb{R}^m$ is the lower limit on the activity for the constraints.
- $u^c \in \mathbb{R}^m$ is the upper limit on the activity for the constraints.
- $l^x \in \mathbb{R}^n$ is the lower limit on the activity for the variables.
- $u^x \in \mathbb{R}^n$ is the upper limit on the activity for the variables.
- $q : \mathbb{R}^n \rightarrow \mathbb{R}$ is a vector of quadratic functions. Hence,

$$q_i(x) = \frac{1}{2} x^T Q^i x$$

where it is assumed that

$$Q^i = (Q^i)^T.$$

Please note the explicit $\frac{1}{2}$ in the quadratic term and that Q^i is required to be symmetric.

- \mathcal{K} is a convex cone.
- $\mathcal{J} \subseteq \{1, 2, \dots, n\}$ is an index set of the integer-constrained variables.

An MPS file with one row and one column can be illustrated like this:

```
*          1          2          3          4          5          6
*23456789012345678901234567890123456789012345678901234567890
NAME          [name]
OBJSENSE
[objsense]
OBJNAME
[objname]
ROWS
? [cname1]
COLUMNS
[vname1] [cname1] [value1] [vname3] [value2]
RHS
[name] [cname1] [value1] [cname2] [value2]
RANGES
[name] [cname1] [value1] [cname2] [value2]
QSECTION      [cname1]
[vname1] [vname2] [value1] [vname3] [value2]
QMATRIX
[vname1] [vname2] [value1]
QUADOBJ
[vname1] [vname2] [value1]
QCMATRIX      [cname1]
[vname1] [vname2] [value1]
BOUNDS
?? [name] [vname1] [value1]
CSECTION      [kname1] [value1] [ktype]
[vname1]
ENDATA
```

Here the names in capitals are keywords of the MPS format and names in brackets are custom defined names or values. A couple of notes on the structure:

- Fields: All items surrounded by brackets appear in *fields*. The fields named “valueN” are numerical values. Hence, they must have the format

```
[+|-]XXXXXXXX.XXXXXX[[e|E][+|-]XXX]
```

where


```
.. 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 19.2.9 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.

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:

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

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_{kj}^i is assigned the value given by [value1]. An optional second element is specified in the same way by the fields [vname1], [vname3], and [value2].

The example

$$\begin{array}{ll} \text{minimize} & -x_2 + \frac{1}{2}(2x_1^2 - 2x_1x_3 + 0.2x_2^2 + 2x_3^2) \\ \text{subject to} & x_1 + x_2 + x_3 \geq 1, \\ & x \geq 0 \end{array}$$

has the following MPS file representation

```
* File: qo1.mps
NAME          qo1
ROWS
N  obj
G  c1
COLUMNS
x1      c1      1.0
x2      obj     -1.0
x2      c1      1.0
x3      c1      1.0
RHS
rhs      c1      1.0
QSECTION      obj
x1      x1      2.0
x1      x3     -1.0
x2      x2      0.2
x3      x3      2.0
ENDATA
```

Regarding the QSECTIONS please note that:

- Only one QSECTION is allowed for each constraint.
- The QSECTIONS can appear in an arbitrary order after the COLUMNS section.
- All variable names occurring in the QSECTION must already be specified in the COLUMNS section.
- All entries specified in a QSECTION are assumed to belong to the lower triangular part of the quadratic term of Q .

QMATRIX/QUADOBJ (optional)

The QMATRIX and QUADOBJ sections allow to define the quadratic term of the objective function. They differ in how the quadratic term of the objective function is stored:

- QMATRIX It stores all the nonzeros coefficients, without 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 appear for each off-diagonal coefficient if using a QMATRIX section, while only one entry is required in a QUADOBJ section. The quadratic part of the objective function will be evaluated as $1/2x^T Qx$.

The example

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

has the following MPS file representation using QMATRIX

```

* File: qo1_matrix.mps
NAME          qo1_qmatrix
ROWS
  N  obj
  G  c1
COLUMNS
  x1      c1      1.0
  x2      obj     -1.0
  x2      c1      1.0
  x3      c1      1.0
RHS
  rhs     c1      1.0
QMATRIX
  x1      x1      2.0
  x1      x3     -1.0
  x3      x1     -1.0
  x2      x2      0.2
  x3      x3      2.0
ENDATA

```

or the following using QUADOBJ

```
* File: qo1_quadobj.mps
NAME          qo1_quadobj
ROWS
  N  obj
  G  c1
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.

19.2.2 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_{kj}^i is assigned the value given by [value1]. Moreover, the quadratic term is represented as $1/2x^T Qx$.

The example

$$\begin{aligned}
 &\text{minimize} && x_2 \\
 &\text{subject to} && x_1 + x_2 + x_3 \geq 1, \\
 & && \frac{1}{2}(-2x_1x_3 + 0.2x_2^2 + 2x_3^2) \leq 10, \\
 & && x \geq 0
 \end{aligned}$$

has the following MPS file representation

```
* File: qo1.mps
NAME          qo1
ROWS
  N  obj
  G  c1
  L  q1
COLUMNS
```

x1	c1	1.0
x2	obj	-1.0
x2	c1	1.0
x3	c1	1.0
RHS		
rhs	c1	1.0
rhs	q1	10.0
QCMATRIX		
	q1	
x1	x1	2.0
x1	x3	-1.0
x3	x1	-1.0
x2	x2	0.2
x3	x3	2.0
ENDATA		

Regarding the QCMATRIXs please note that:

- Only one QCMATRIX is allowed for each constraint.
- The QCMATRIXs can appear in an arbitrary order after the COLUMNS section.
- All variable names occurring in the QSECTION must already be specified in the COLUMNS section.
- A QCMATRIX does not exploit the symmetry of Q : an off-diagonal entry (i, j) should appear twice.

19.2.3 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].

19.2.4 CSECTION (optional)

The purpose of the CSECTION is to specify the constraint

$$x \in \mathcal{K}.$$

in (19.2). It is assumed that \mathcal{K} satisfies the following requirements. Let

$$x^t \in \mathbb{R}^{n^t}, \quad t = 1, \dots, k$$

be vectors comprised of parts of the decision variables x so that each decision variable is a member of exactly **one** vector x^t , for example

$$x^1 = \begin{bmatrix} x_1 \\ x_4 \\ x_7 \end{bmatrix} \quad \text{and} \quad x^2 = \begin{bmatrix} x_6 \\ x_5 \\ x_3 \\ x_2 \end{bmatrix}.$$

Next define

$$\mathcal{K} := \{x \in \mathbb{R}^n : x^t \in \mathcal{K}_t, \quad t = 1, \dots, k\}$$

where \mathcal{K}_t must have one of the following forms

- \mathbb{R} set:

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

- Quadratic cone:

$$\mathcal{K}_t = \left\{ x \in \mathbb{R}^{n^t} : x_1 \geq \sqrt{\sum_{j=2}^{n^t} x_j^2} \right\}. \quad (19.3)$$

- Rotated quadratic cone:

$$\mathcal{K}_t = \left\{ x \in \mathbb{R}^{n^t} : 2x_1x_2 \geq \sum_{j=3}^{n^t} x_j^2, \quad x_1, x_2 \geq 0 \right\}. \quad (19.4)$$

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

$$x_3x_7 \geq x_1^2 + x_0^2, \quad x_3, x_7 \geq 0,$$

should be specified in the MPS file. One CSECTION is required for each cone and they are specified as follows:

*	1	2	3	4	5	6
*234567890123456789012345678901234567890123456789012345678901234567890						
CSECTION	konea	0.0	QUAD			
x4						
x5						
x8						
CSECTION	koneb	0.0	RQUAD			
x7						
x3						
x1						
x0						

This first CSECTION specifies the cone (19.3) 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 (19.4). 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. (19.3).
RQUAD	≤ 2	Rotated quadratic cone i.e. (19.4).

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.

19.2.5 ENDATA

This keyword denotes the end of the MPS file.

19.2.6 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:

```

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.

19.2.7 General Limitations

- An MPS file should be an ASCII file.

19.2.8 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.

19.2.9 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.

19.3 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.

19.3.1 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]
```

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.

- A quadratic cone of n variables x_1, \dots, x_n defines a constraint of the form

$$x_1^2 \geq \sum_{i=2}^n x_i^2, \quad x_1 \geq 0.$$

- A rotated quadratic cone of n variables x_1, \dots, x_n defines a constraint of the form

$$2x_1x_2 \geq \sum_{i=3}^n x_i^2, \quad x_1, x_2 \geq 0.$$

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. Optionally, an attribute can be added [variables disallow_new_variables] indicating that if any variable not listed here occurs later in the file it is an error.

[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.

```
[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.
- 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.
- **s1** Defines the level of the dual variable associated with its lower bound.
- **su** Defines the level of the dual variable associated with its upper bound.
- **sn** Defines the level of the variable associated with its cone.
- **y** Defines the level of the corresponding dual variable (for constraints only).

A **[var]** section should always contain the items **sk**, **lvl**, **s1** and **su**. Items **s1** and **su** are not required for **integer** solutions.

A **[con]** section should always contain **sk**, **lvl**, **s1**, **su** and **y**.

An example of a solution section

```
[solution basic status=UNKNOWN]
[var x0] sk=LOW    lvl=5.0      [/var]
[var x1] sk=UPR    lvl=10.0     [/var]
[var x2] sk=SUPBAS lvl=2.0    s1=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]+)?
```

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"
```

19.3.2 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]
```

19.3.3 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:

<i>iparam.opf_write_sol_bas</i>	Include basic solution, if defined.
<i>iparam.opf_write_sol_itg</i>	Include integer solution, if defined.
<i>iparam.opf_write_sol_itr</i>	Include interior solution, if defined.
<i>iparam.opf_write_solutions</i>	Include solutions if they are defined. If this is off, no solutions are included.
<i>iparam.opf_write_header</i>	Include a small header with comments.
<i>iparam.opf_write_problem</i>	Include the problem itself — objective, constraints and bounds.
<i>iparam.opf_write_parameters</i>	Include all parameter settings.
<i>iparam.opf_write_hints</i>	Include hints about the size of the problem.

19.3.4 Examples

This section contains a set of small examples written in OPF and describing how to formulate linear, quadratic and conic problems.

Linear Example lo1.opf

Consider the example:

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

having the bounds

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

In the OPF format the example is displayed as shown in [Listing 19.1](#).

Listing 19.1: 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

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

This can be formulated in `opf` as shown below.

Listing 19.2: 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:

$$\begin{aligned}
 &\text{minimize} && x_3 + x_4 + x_5 \\
 &\text{subject to} && x_0 + x_1 + 2x_2 = 1, \\
 & && x_0, x_1, x_2 \geq 0, \\
 & && x_3 \geq \sqrt{x_0^2 + x_1^2}, \\
 & && 2x_4x_5 \geq x_2^2.
 \end{aligned}$$

Please note that the type of the cones is defined by the parameter to `[cone ...]`; the content of the cone-section is the names of variables that belong to the cone. The resulting OPF file is in [Listing 19.3](#).

Listing 19.3: Example of an OPF file for a conic quadratic problem.

```
[comment]
  The cqo1 example in OPF format.
[/comment]

[hints]
  [hint NUMVAR] 6 [/hint]
  [hint NUMCON] 1 [/hint]
  [hint NUMANZ] 3 [/hint]
[/hints]

[variables disallow_new_variables]
  x1 x2 x3 x4 x5 x6
[/variables]
```

```

[objective minimize 'obj']
  x4 + x5 + x6
[/objective]

[constraints]
  [con 'c1']  x1 + x2 + 2e+00 x3 = 1e+00 [/con]
[/constraints]

[bounds]
  # We let all variables default to the positive orthant
  [b] 0 <= * [/b]

  # ...and change those that differ from the default
  [b] x4,x5,x6 free [/b]

  # Define quadratic cone:  $x_4 \geq \sqrt{x_1^2 + x_2^2}$ 
  [cone quad 'k1'] x4, x1, x2 [/cone]

  # Define rotated quadratic cone:  $2 x_5 x_6 \geq x_3^2$ 
  [cone rquad 'k2'] x5, x6, x3 [/cone]
[/bounds]

```

Mixed Integer Example `mil01.opf`

Consider the mixed integer problem:

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

This can be implemented in OPF with the file in [Listing 19.4](#).

Listing 19.4: Example of an OPF file for a mixed-integer linear problem.

```

[comment]
  The mil01 example in OPF format
[/comment]

[hints]
  [hint NUMVAR] 2 [/hint]
  [hint NUMCON] 2 [/hint]
  [hint NUMANZ] 4 [/hint]
[/hints]

[variables disallow_new_variables]
  x1 x2
[/variables]

[objective maximize 'obj']
  x1 + 6.4e-1 x2
[/objective]

[constraints]
  [con 'c1'] 5e+1 x1 + 3.1e+1 x2 <= 2.5e+2 [/con]
  [con 'c2'] -4 <= 3 x1 - 2 x2 [/con]
[/constraints]

[bounds]
  [b] 0 <= * [/b]
[/bounds]

```

```
[integer]
  x1 x2
[/integer]
```

19.4 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.

19.4.1 How Instances Are Specified

This section defines the spectrum of conic optimization problems that can be formulated in terms of the keywords of the CBF format.

In the CBF format, conic optimization problems are considered in the following form:

$$\begin{aligned}
 & \min / \max && g^{obj} \\
 & \text{s.t.} && \begin{aligned} & g_i \in \mathcal{K}_i, & i \in \mathcal{I}, \\ & G_i \in \mathcal{K}_i, & i \in \mathcal{I}^{PSD}, \\ & x_j \in \mathcal{K}_j, & j \in \mathcal{J}, \\ & \bar{X}_j \in \mathcal{K}_j, & j \in \mathcal{J}^{PSD}. \end{aligned}
 \end{aligned} \tag{19.5}$$

- **Variables** are either scalar variables, x_j for $j \in \mathcal{J}$, or variables, \bar{X}_j for $j \in \mathcal{J}^{PSD}$. Scalar variables can also be declared as integer.
- **Constraints** are affine expressions of the variables, either scalar-valued g_i for $i \in \mathcal{I}$, or matrix-valued G_i for $i \in \mathcal{I}^{PSD}$

$$\begin{aligned}
 g_i &= \sum_{j \in \mathcal{J}^{PSD}} \langle F_{ij}, X_j \rangle + \sum_{j \in \mathcal{J}} a_{ij} x_j + b_i, \\
 G_i &= \sum_{j \in \mathcal{J}} x_j H_{ij} + D_i.
 \end{aligned}$$

- The **objective function** is a scalar-valued affine expression of the variables, either to be minimized or maximized. We refer to this expression as g^{obj}

$$g^{obj} = \sum_{j \in \mathcal{J}^{PSD}} \langle F_j^{obj}, X_j \rangle + \sum_{j \in \mathcal{J}} a_j^{obj} x_j + b^{obj}.$$

CBF format can represent the following cones \mathcal{K} :

- **Free domain** - A cone in the linear family defined by

$$\{x \in \mathbb{R}^n\}, \text{ for } n \geq 1.$$

- **Positive orthant** - A cone in the linear family defined by

$$\{x \in \mathbb{R}^n \mid x_j \geq 0 \text{ for } j = 1, \dots, n\}, \text{ for } n \geq 1.$$

- **Negative orthant** - A cone in the linear family defined by

$$\{x \in \mathbb{R}^n \mid x_j \leq 0 \text{ for } j = 1, \dots, n\}, \text{ for } n \geq 1.$$

- **Fixpoint zero** - A cone in the linear family defined by

$$\{x \in \mathbb{R}^n \mid x_j = 0 \text{ for } j = 1, \dots, n\}, \text{ for } n \geq 1.$$

- **Quadratic cone** - A cone in the second-order cone family defined by

$$\left\{ \begin{pmatrix} p \\ x \end{pmatrix} \in \mathbb{R} \times \mathbb{R}^{n-1}, p^2 \geq x^T x, p \geq 0 \right\}, \text{ for } n \geq 2.$$

- **Rotated quadratic cone** - A cone in the second-order cone family defined by

$$\left\{ \begin{pmatrix} p \\ q \\ x \end{pmatrix} \in \mathbb{R} \times \mathbb{R} \times \mathbb{R}^{n-2}, 2pq \geq x^T x, p \geq 0, q \geq 0 \right\}, \text{ for } n \geq 3.$$

19.4.2 The Structure of CBF Files

This section defines how information is written in the CBF format, without being specific about the type of information being communicated.

All information items belong to exactly one of the three groups of information. These information groups, and the order they must appear in, are:

1. File format.
2. Problem structure.
3. Problem data.

The first group, file format, provides information on how to interpret the file. The second group, problem structure, provides the information needed to deduce the type and size of the problem instance. Finally, the third group, problem data, specifies the coefficients and constants of the problem instance.

Information items

The format is composed as a list of information items. The first line of an information item is the **KEYWORD**, revealing the type of information provided. The second line - of some keywords only - is the **HEADER**, typically revealing the size of information that follows. The remaining lines are the **BODY** holding the actual information to be specified.

KEYWORD
BODY
KEYWORD
HEADER
BODY

The **KEYWORD** determines how each line in the **HEADER** and **BODY** is structured. Moreover, the number of lines in the **BODY** follows either from the **KEYWORD**, the **HEADER**, or from another information item required to precede it.

Embedded hotstart-sequences

A sequence of problem instances, based on the same problem structure, is within a single file. This is facilitated via the **CHANGE** within the problem data information group, as a separator between the information items of each instance. The information items following a **CHANGE** keyword are appending to, or changing (e.g., setting coefficients back to their default value of zero), the problem data of the preceding instance.

The sequence is intended for benchmarking of hotstart capability, where the solvers can reuse their internal state and solution (subject to the achieved accuracy) as warmpoint for the succeeding instance. Whenever this feature is unsupported or undesired, the keyword **CHANGE** should be interpreted as the end of file.

File encoding and line width restrictions

The format is based on the US-ASCII printable character set with two extensions as listed below. Note, by definition, that none of these extensions can be misinterpreted as printable US-ASCII characters:

- A line feed marks the end of a line, carriage returns are ignored.
- Comment-lines may contain unicode characters in UTF-8 encoding.

The line width is restricted to 512 bytes, with 3 bytes reserved for the potential carriage return, line feed and null-terminator.

Integers and floating point numbers must follow the ISO C decimal string representation in the standard C locale. The format does not impose restrictions on the magnitude of, or number of significant digits in numeric data, but the use of 64-bit integers and 64-bit IEEE 754 floating point numbers should be sufficient to avoid loss of precision.

Comment-line and whitespace rules

The format allows single-line comments respecting the following rule:

- Lines having first byte equal to '#' (US-ASCII 35) are comments, and should be ignored. Comments are only allowed between information items.

Given that a line is not a comment-line, whitespace characters should be handled according to the following rules:

- Leading and trailing whitespace characters should be ignored.
 - The separator between multiple pieces of information on one line, is either one or more whitespace characters.
- Lines containing only whitespace characters are empty, and should be ignored. Empty lines are only allowed between information items.

19.4.3 Problem Specification

The problem structure

The problem structure defines the objective sense, whether it is minimization and maximization. It also defines the index sets, \mathcal{J} , \mathcal{J}^{PSD} , \mathcal{I} and \mathcal{I}^{PSD} , which are all numbered from zero, $\{0, 1, \dots\}$, and empty until explicitly constructed.

- **Scalar variables** are constructed in vectors restricted to a conic domain, such as $(x_0, x_1) \in \mathbb{R}_+^2$, $(x_2, x_3, x_4) \in \mathcal{Q}^3$, etc. In terms of the Cartesian product, this generalizes to

$$x \in \mathcal{K}_1^{n_1} \times \mathcal{K}_2^{n_2} \times \dots \times \mathcal{K}_k^{n_k}$$

which in the CBF format becomes:

```
VAR
n k
K1 n1
K2 n2
...
Kk nk
```

where $\sum_i n_i = n$ is the total number of scalar variables. The list of supported cones is found in [Table 19.3](#). Integrality of scalar variables can be specified afterwards.

- **PSD variables** are constructed one-by-one. That is, $X_j \succeq \mathbf{0}^{n_j \times n_j}$ for $j \in \mathcal{J}^{PSD}$, constructs a matrix-valued variable of size $n_j \times n_j$ restricted to be symmetric positive semidefinite. In the CBF format, this list of constructions becomes:

```
PSDVAR
N
n1
n2
...
nN
```

where N is the total number of PSD variables.

- **Scalar constraints** are constructed in vectors restricted to a conic domain, such as $(g_0, g_1) \in \mathbb{R}_+^2$, $(g_2, g_3, g_4) \in \mathcal{Q}^3$, etc. In terms of the Cartesian product, this generalizes to

$$g \in \mathcal{K}_1^{m_1} \times \mathcal{K}_2^{m_2} \times \dots \times \mathcal{K}_k^{m_k}$$

which in the CBF format becomes:

```
CON
m k
K1 m1
K2 m2
..
Kk mk
```

where $\sum_i m_i = m$ is the total number of scalar constraints. The list of supported cones is found in [Table 19.3](#).

- **PSD constraints** are constructed one-by-one. That is, $G_i \succeq \mathbf{0}^{m_i \times m_i}$ for $i \in \mathcal{I}^{PSD}$, constructs a matrix-valued affine expressions of size $m_i \times m_i$ restricted to be symmetric positive semidefinite. In the CBF format, this list of constructions becomes

```
PSDCON
M
m1
m2
..
mM
```

where M is the total number of PSD constraints.

With the objective sense, variables (with integer indications) and constraints, the definitions of the many affine expressions follow in problem data.

Problem data

The problem data defines the coefficients and constants of the affine expressions of the problem instance. These are considered zero until explicitly defined, implying that instances with no keywords from this

information group are, in fact, valid. Duplicating or conflicting information is a failure to comply with the standard. Consequently, two coefficients written to the same position in a matrix (or to transposed positions in a symmetric matrix) is an error.

The affine expressions of the objective, g^{obj} , of the scalar constraints, g_i , and of the PSD constraints, G_i , are defined separately. The following notation uses the standard trace inner product for matrices, $\langle X, Y \rangle = \sum_{i,j} X_{ij}Y_{ij}$.

- The affine expression of the objective is defined as

$$g^{obj} = \sum_{j \in \mathcal{J}^{PSD}} \langle F_j^{obj}, X_j \rangle + \sum_{j \in \mathcal{J}} a_j^{obj} x_j + b^{obj},$$

in terms of the symmetric matrices, F_j^{obj} , and scalars, a_j^{obj} and b^{obj} .

- The affine expressions of the scalar constraints are defined, for $i \in \mathcal{I}$, as

$$g_i = \sum_{j \in \mathcal{J}^{PSD}} \langle F_{ij}, X_j \rangle + \sum_{j \in \mathcal{J}} a_{ij} x_j + b_i,$$

in terms of the symmetric matrices, F_{ij} , and scalars, a_{ij} and b_i .

- The affine expressions of the PSD constraints are defined, for $i \in \mathcal{I}^{PSD}$, as

$$G_i = \sum_{j \in \mathcal{J}} x_j H_{ij} + D_i,$$

in terms of the symmetric matrices, H_{ij} and D_i .

List of cones

The format uses an explicit syntax for symmetric positive semidefinite cones as shown above. For scalar variables and constraints, constructed in vectors, the supported conic domains and their minimum sizes are given as follows.

Table 19.3: 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

19.4.4 File Format Keywords

VER

Description: The version of the Conic Benchmark Format used to write the file.

HEADER: None

BODY: One line formatted as:

INT

This is the version number.

Must appear exactly once in a file, as the first keyword.

OBJSENSE

Description: Define the objective sense.

HEADER: None

BODY: One line formatted as:

STR

having MIN indicates minimize, and MAX indicates maximize. Capital letters are required.

Must appear exactly once in a file.

PSDVAR

Description: Construct the PSD variables.

HEADER: One line formatted as:

INT

This is the number of PSD variables in the problem.

BODY: A list of lines formatted as:

INT

This indicates the number of rows (equal to the number of columns) in the matrix-valued PSD variable. The number of lines should match the number stated in the header.

VAR

Description: Construct the scalar variables.

HEADER: One line formatted as:

INT INT

This is the number of scalar variables, followed by the number of conic domains they are restricted to.

BODY: A list of lines formatted as:

STR INT

This indicates the cone name (see [Table 19.3](#)), and the number of scalar variables restricted to this cone. These numbers should add up to the number of scalar variables stated first in the header. The number of lines should match the second number stated in the header.

INT

Description: Declare integer requirements on a selected subset of scalar variables.

HEADER: one line formatted as:

INT

This is the number of integer scalar variables in the problem.

BODY: a list of lines formatted as:

INT

This indicates the scalar variable index $j \in \mathcal{J}$. The number of lines should match the number stated in the header.

Can only be used after the keyword **VAR**.

PSDCON

Description: Construct the PSD constraints.

HEADER: One line formatted as:

INT

This is the number of PSD constraints in the problem.

BODY: A list of lines formatted as:

INT

This indicates the number of rows (equal to the number of columns) in the matrix-valued affine expression of the PSD constraint. The number of lines should match the number stated in the header.

Can only be used after these keywords: **PSDVAR**, **VAR**.

CON

Description: Construct the scalar constraints.

HEADER: One line formatted as:

INT INT

This is the number of scalar constraints, followed by the number of conic domains they restrict to.

BODY: A list of lines formatted as:

STR INT

This indicates the cone name (see [Table 19.3](#)), and the number of affine expressions restricted to this cone. These numbers should add up to the number of scalar constraints stated first in the header. The number of lines should match the second number stated in the header.

Can only be used after these keywords: **PSDVAR**, **VAR**.

OBJFCOORD

Description: Input sparse coordinates (quadruplets) to define the symmetric matrices F_j^{obj} , as used in the objective.

HEADER: One line formatted as:

INT

This is the number of coordinates to be specified.

BODY: A list of lines formatted as:

INT INT INT REAL

This indicates the PSD variable index $j \in \mathcal{J}^{PSD}$, the row index, the column index and the coefficient value. The number of lines should match the number stated in the header.

OBJACOORD

Description: Input sparse coordinates (pairs) to define the scalars, a_j^{obj} , as used in the objective.

HEADER: One line formatted as:

INT

This is the number of coordinates to be specified.

BODY: A list of lines formatted as:

INT REAL

This indicates the scalar variable index $j \in \mathcal{J}$ and the coefficient value. The number of lines should match the number stated in the header.

OBJBCOORD

Description: Input the scalar, b^{obj} , as used in the objective.

HEADER: None.

BODY: One line formatted as:

REAL

This indicates the coefficient value.

FCOORD

Description: Input sparse coordinates (quintuplets) to define the symmetric matrices, F_{ij} , as used in the scalar constraints.

HEADER: One line formatted as:

INT

This is the number of coordinates to be specified.

BODY: A list of lines formatted as:

INT INT INT INT REAL

This indicates the scalar constraint index $i \in \mathcal{I}$, the PSD variable index $j \in \mathcal{J}^{PSD}$, the row index, the column index and the coefficient value. The number of lines should match the number stated in the header.

ACOORD

Description: Input sparse coordinates (triplets) to define the scalars, a_{ij} , as used in the scalar constraints.

HEADER: One line formatted as:

INT

This is the number of coordinates to be specified.

BODY: A list of lines formatted as:

INT INT REAL

This indicates the scalar constraint index $i \in \mathcal{I}$, the scalar variable index $j \in \mathcal{J}$ and the coefficient value. The number of lines should match the number stated in the header.

BCOORD

Description: Input sparse coordinates (pairs) to define the scalars, b_i , as used in the scalar constraints.

HEADER: One line formatted as:

INT

This is the number of coordinates to be specified.

BODY: A list of lines formatted as:

INT REAL

This indicates the scalar constraint index $i \in \mathcal{I}$ and the coefficient value. The number of lines should match the number stated in the header.

HCOORD

Description: Input sparse coordinates (quintuplets) to define the symmetric matrices, H_{ij} , as used in the PSD constraints.

HEADER: One line formatted as:

INT

This is the number of coordinates to be specified.

BODY: A list of lines formatted as

INT INT INT INT REAL

This indicates the PSD constraint index $i \in \mathcal{I}^{PSD}$, the scalar variable index $j \in \mathcal{J}$, the row index, the column index and the coefficient value. The number of lines should match the number stated in the header.

DCOORD

Description: Input sparse coordinates (quadruplets) to define the symmetric matrices, D_i , as used in the PSD constraints.

HEADER: One line formatted as

INT

This is the number of coordinates to be specified.

BODY: A list of lines formatted as:

INT INT INT REAL

This indicates the PSD constraint index $i \in \mathcal{I}^{PSD}$, the row index, the column index and the coefficient value. The number of lines should match the number stated in the header.

CHANGE

Start of a new instance specification based on changes to the previous. Can be interpreted as the end of file when the hotstart-sequence is unsupported or undesired.

BODY: None

Header: None

19.4.5 CBF Format Examples

Minimal Working Example

The conic optimization problem (19.6), has three variables in a quadratic cone - first one is integer - and an affine expression in domain 0 (equality constraint).

$$\begin{array}{ll} \text{minimize} & 5.1 x_0 \\ \text{subject to} & 6.2 x_1 + 7.3 x_2 - 8.4 \in \{0\} \\ & x \in \mathcal{Q}^3, x_0 \in \mathbb{Z}. \end{array} \quad (19.6)$$

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.

Mixing Linear, Second-order and Semidefinite Cones

The conic optimization problem (19.7), has a semidefinite cone, a quadratic cone over unordered subindices, and two equality constraints.

$$\begin{aligned}
 & \text{minimize} && \left\langle \begin{bmatrix} 2 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 2 \end{bmatrix}, X_1 \right\rangle + x_1 \\
 & \text{subject to} && \left\langle \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, X_1 \right\rangle + x_1 &= 1.0, \\
 & && \left\langle \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, X_1 \right\rangle + x_0 + x_2 &= 0.5, \\
 & && x_1 \geq \sqrt{x_0^2 + x_2^2}, \\
 & && X_1 \succeq \mathbf{0}.
 \end{aligned} \tag{19.7}$$

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

```

```

0 2 1 1.0
0 2 2 2.0

# One coordinate in a^{obj}_j coefficients:
#   | a^{obj}[1] = 1.0
OBJCOORD
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...
ACCOORD
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.

$$\begin{aligned}
 & \text{minimize} && \left\langle \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, X_1 \right\rangle + x_1 + x_2 + 1 \\
 & \text{subject to} && \left\langle \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, X_1 \right\rangle - x_1 - x_2 \geq 0.0, \\
 & && x_1 \begin{bmatrix} 0 & 1 \\ 1 & 3 \end{bmatrix} + x_2 \begin{bmatrix} 3 & 1 \\ 1 & 0 \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \succeq \mathbf{0}, \\
 & && X_1 \succeq \mathbf{0}.
 \end{aligned} \tag{19.8}$$

Its formulation in the CBF format is written in what follows

```

# File written using this version of the Conic Benchmark Format:
#   | Version 1.
VER
1

# The sense of the objective is:
#   | Minimize.
OBJSENSE
MIN

# One PSD variable of this size:
#   | Two times two.
PSDVAR
1
2

# Two scalar variables in this one conic domain:
#   | Two are free.
VAR
2 1
F 2

# One PSD constraint of this size:
#   | Two times two.
PSDCON
1
2

# One scalar constraint with an affine expression in this one conic domain:
#   | One is greater than or equal to zero.
CON
1 1
L+ 1

# Two coordinates in  $F^{\{obj\}}_j$  coefficients:
#   |  $F^{\{obj\}}[0][0,0] = 1.0$ 
#   |  $F^{\{obj\}}[0][1,1] = 1.0$ 
OBJFCOORD
2
0 0 0 1.0
0 1 1 1.0

# Two coordinates in  $a^{\{obj\}}_j$  coefficients:
#   |  $a^{\{obj\}}[0] = 1.0$ 
#   |  $a^{\{obj\}}[1] = 1.0$ 
OBJACOORD
2
0 1.0
1 1.0

# One coordinate in  $b^{\{obj\}}$  coefficient:
#   |  $b^{\{obj\}} = 1.0$ 
OBJBCOORD
1.0

# One coordinate in  $F_{ij}$  coefficients:
#   |  $F[0,0][1,0] = 1.0$ 
FCOORD
1
0 0 1 0 1.0

# Two coordinates in  $a_{ij}$  coefficients:
#   |  $a[0,0] = -1.0$ 

```



```
#      | a[0,1] = -1.0
ACCOORD
2
0 0 -1.0
0 1 -1.0

# Four coordinates in H_ij coefficients:
#      | H[0,0][1,0] = 1.0
#      | H[0,0][1,1] = 3.0
#      | and more...
HCOORD
4
0 0 1 0 1.0
0 0 1 1 3.0
0 1 0 0 3.0
0 1 1 0 1.0

# Two coordinates in D_i coefficients:
#      | D[0][0,0] = -1.0
#      | D[0][1,1] = -1.0
DCCOORD
2
0 0 0 -1.0
0 1 1 -1.0
```

Optimization Over a Sequence of Objectives

The linear optimization problem (19.9), is defined for a sequence of objectives such that hotstarting from one to the next might be advantages.

$$\begin{aligned}
 & \text{maximize}_k && g_k^{obj} \\
 & \text{subject to} && 50x_0 + 31 \leq 250, \\
 & && 3x_0 - 2x_1 \geq -4, \\
 & && x \in \mathbb{R}_+^2,
 \end{aligned} \tag{19.9}$$

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 19.5](#).

Listing 19.5: Problem (19.9) 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
```

```
# 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...
ACCOORD
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
BCCOORD
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
```

19.5 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.

19.6 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.

19.7 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.

19.7.1 *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.

19.7.2 *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.

19.7.3 A *jtask* example

In Listing 19.6 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 19.6: 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"
      ],
      "b1": [
        0.0,
        0.0,
        0.0,
        0.0
      ],
      "bu": [
        1e+30,
        1e+1,
        1e+30,

```

```

        1e+30
    ],
    "type": [
        "cont",
        "cont",
        "cont",
        "cont"
    ]
},
"con": {
    "name": [
        "c1",
        "c2",
        "c3"
    ],
    "bk": [
        "fx",
        "lo",
        "up"
    ],
    "bl": [
        3e+1,
        1.5e+1,
        -1e+30
    ],
    "bu": [
        3e+1,
        1e+30,
        2.5e+1
    ]
},
"objective": {
    "sense": "max",
    "name": "obj",
    "c": {
        "subj": [
            0,
            1,
            2,
            3
        ],
        "val": [
            3e+0,
            1e+0,
            5e+0,
            1e+0
        ]
    }
},
"cfix": 0.0
},
"A": {
    "subi": [
        0,
        0,
        0,
        1,
        1,
        1,
        1,
        1,
        2,
        2
    ],
    "subj": [

```

```

        0,
        1,
        2,
        0,
        1,
        2,
        3,
        1,
        3
    ],
    "val": [
        3e+0,
        1e+0,
        2e+0,
        2e+0,
        1e+0,
        3e+0,
        1e+0,
        2e+0,
        3e+0
    ]
}
},
"Task/parameters": {
    "iparam": {
        "ANA_SOL_BASIS": "ON",
        "ANA_SOL_PRINT_VIOLATED": "OFF",
        "AUTO_SORT_A_BEFORE_OPT": "OFF",
        "AUTO_UPDATE_SOL_INFO": "OFF",
        "BASIS_SOLVE_USE_PLUS_ONE": "OFF",
        "BI_CLEAN_OPTIMIZER": "OPTIMIZER_FREE",
        "BI_IGNORE_MAX_ITER": "OFF",
        "BI_IGNORE_NUM_ERROR": "OFF",
        "BI_MAX_ITERATIONS": 1000000,
        "CACHE_LICENSE": "ON",
        "CHECK_CONVEXITY": "CHECK_CONVEXITY_FULL",
        "COMPRESS_STATFILE": "ON",
        "CONCURRENT_NUM_OPTIMIZERS": 2,
        "CONCURRENT_PRIORITY_DUAL_SIMPLEX": 2,
        "CONCURRENT_PRIORITY_FREE_SIMPLEX": 3,
        "CONCURRENT_PRIORITY_INTPNT": 4,
        "CONCURRENT_PRIORITY_PRIMAL_SIMPLEX": 1,
        "FEASREPAIR_OPTIMIZE": "FEASREPAIR_OPTIMIZE_NONE",
        "INFEAS_GENERIC_NAMES": "OFF",
        "INFEAS_PREFER_PRIMAL": "ON",
        "INFEAS_REPORT_AUTO": "OFF",
        "INFEAS_REPORT_LEVEL": 1,
        "INTPNT_BASIS": "BI_ALWAYS",
        "INTPNT_DIFF_STEP": "ON",
        "INTPNT_FACTOR_DEBUG_LVL": 0,
        "INTPNT_FACTOR_METHOD": 0,
        "INTPNT_HOTSTART": "INTPNT_HOTSTART_NONE",
        "INTPNT_MAX_ITERATIONS": 400,
        "INTPNT_MAX_NUM_COR": -1,
        "INTPNT_MAX_NUM_REFINEMENT_STEPS": -1,
        "INTPNT_OFF_COL_TRH": 40,
        "INTPNT_ORDER_METHOD": "ORDER_METHOD_FREE",
        "INTPNT_REGULARIZATION_USE": "ON",
        "INTPNT_SCALING": "SCALING_FREE",
        "INTPNT_SOLVE_FORM": "SOLVE_FREE",
        "INTPNT_STARTING_POINT": "STARTING_POINT_FREE",
        "LIC_TRH_EXPIRY_WRN": 7,
        "LICENSE_DEBUG": "OFF",
    }
}

```

```

"LICENSE_PAUSE_TIME":0,
"LICENSE_SUPPRESS_EXPIRE_WRNS":"OFF",
"LICENSE_WAIT":"OFF",
"LOG":10,
"LOG_ANA_PRO":1,
"LOG_BI":4,
"LOG_BI_FREQ":2500,
"LOG_CHECK_CONVEXITY":0,
"LOG_CONCURRENT":1,
"LOG_CUT_SECOND_OPT":1,
"LOG_EXPAND":0,
"LOG_FACTOR":1,
"LOG_FEAS_REPAIR":1,
"LOG_FILE":1,
"LOG_HEAD":1,
"LOG_INFEAS_ANA":1,
"LOG_INTPNT":4,
"LOG_MIO":4,
"LOG_MIO_FREQ":1000,
"LOG_OPTIMIZER":1,
"LOG_ORDER":1,
"LOG_PRESOLVE":1,
"LOG_RESPONSE":0,
"LOG_SENSITIVITY":1,
"LOG_SENSITIVITY_OPT":0,
"LOG_SIM":4,
"LOG_SIM_FREQ":1000,
"LOG_SIM_MINOR":1,
"LOG_STORAGE":1,
"MAX_NUM_WARNINGS":10,
"MIO_BRANCH_DIR":"BRANCH_DIR_FREE",
"MIO_CONSTRUCT_SOL":"OFF",
"MIO_CUT_CLIQUE":"ON",
"MIO_CUT_CMIR":"ON",
"MIO_CUT_GMI":"ON",
"MIO_CUT_KNAPSACK_COVER":"OFF",
"MIO_HEURISTIC_LEVEL":-1,
"MIO_MAX_NUM_BRANCHES":-1,
"MIO_MAX_NUM_RELAXS":-1,
"MIO_MAX_NUM_SOLUTIONS":-1,
"MIO_MODE":"MIO_MODE_SATISFIED",
"MIO_MT_USER_CB":"ON",
"MIO_NODE_OPTIMIZER":"OPTIMIZER_FREE",
"MIO_NODE_SELECTION":"MIO_NODE_SELECTION_FREE",
"MIO_PERSPECTIVE_REFORMULATE":"ON",
"MIO_PROBING_LEVEL":-1,
"MIO_RINS_MAX_NODES":-1,
"MIO_ROOT_OPTIMIZER":"OPTIMIZER_FREE",
"MIO_ROOT_REPEAT_PRESOLVE_LEVEL":-1,
"MT_SPINCOUNT":0,
"NUM_THREADS":0,
"OPF_MAX_TERMS_PER_LINE":5,
"OPF_WRITE_HEADER":"ON",
"OPF_WRITE_HINTS":"ON",
"OPF_WRITE_PARAMETERS":"OFF",
"OPF_WRITE_PROBLEM":"ON",
"OPF_WRITE_SOL_BAS":"ON",
"OPF_WRITE_SOL_ITG":"ON",
"OPF_WRITE_SOL_ITR":"ON",
"OPF_WRITE_SOLUTIONS":"OFF",
"OPTIMIZER":"OPTIMIZER_FREE",
"PARAM_READ_CASE_NAME":"ON",
"PARAM_READ_IGN_ERROR":"OFF",

```

```

"PRESOLVE_ELIMINATOR_MAX_FILL":-1,
"PRESOLVE_ELIMINATOR_MAX_NUM_TRIES":-1,
"PRESOLVE_LEVEL":-1,
"PRESOLVE_LINDEP_ABS_WORK_TRH":100,
"PRESOLVE_LINDEP_REL_WORK_TRH":100,
"PRESOLVE_LINDEP_USE":"ON",
"PRESOLVE_MAX_NUM_REDUCTIONS":-1,
"PRESOLVE_USE":"PRESOLVE_MODE_FREE",
"PRIMAL_REPAIR_OPTIMIZER":"OPTIMIZER_FREE",
"QO_SEPARABLE_REFORMULATION":"OFF",
"READ_DATA_COMPRESSED":"COMPRESS_FREE",
"READ_DATA_FORMAT":"DATA_FORMAT_EXTENSION",
"READ_DEBUG":"OFF",
"READ_KEEP_FREE_CON":"OFF",
"READ_LP_DROP_NEW_VARS_IN_BOU":"OFF",
"READ_LP_QUOTED_NAMES":"ON",
"READ_MPS_FORMAT":"MPS_FORMAT_FREE",
"READ_MPS_WIDTH":1024,
"READ_TASK_IGNORE_PARAM":"OFF",
"SENSITIVITY_ALL":"OFF",
"SENSITIVITY_OPTIMIZER":"OPTIMIZER_FREE_SIMPLEX",
"SENSITIVITY_TYPE":"SENSITIVITY_TYPE_BASIS",
"SIM_BASIS_FACTOR_USE":"ON",
"SIM_DEGEN":"SIM_DEGEN_FREE",
"SIM_DUAL_CRASH":90,
"SIM_DUAL_PHASEONE_METHOD":0,
"SIM_DUAL_RESTRICT_SELECTION":50,
"SIM_DUAL_SELECTION":"SIM_SELECTION_FREE",
"SIM_EXPLOIT_DUPVEC":"SIM_EXPLOIT_DUPVEC_OFF",
"SIM_HOTSTART":"SIM_HOTSTART_FREE",
"SIM_HOTSTART_LU":"ON",
"SIM_INTEGER":0,
"SIM_MAX_ITERATIONS":10000000,
"SIM_MAX_NUM_SETBACKS":250,
"SIM_NON_SINGULAR":"ON",
"SIM_PRIMAL_CRASH":90,
"SIM_PRIMAL_PHASEONE_METHOD":0,
"SIM_PRIMAL_RESTRICT_SELECTION":50,
"SIM_PRIMAL_SELECTION":"SIM_SELECTION_FREE",
"SIM_REFACTOR_FREQ":0,
"SIM_REFORMULATION":"SIM_REFORMULATION_OFF",
"SIM_SAVE_LU":"OFF",
"SIM_SCALING":"SCALING_FREE",
"SIM_SCALING_METHOD":"SCALING_METHOD_POW2",
"SIM_SOLVE_FORM":"SOLVE_FREE",
"SIM_STABILITY_PRIORITY":50,
"SIM_SWITCH_OPTIMIZER":"OFF",
"SOL_FILTER_KEEP_BASIC":"OFF",
"SOL_FILTER_KEEP_RANGED":"OFF",
"SOL_READ_NAME_WIDTH":-1,
"SOL_READ_WIDTH":1024,
"SOLUTION_CALLBACK":"OFF",
"TIMING_LEVEL":1,
"WRITE_BAS_CONSTRAINTS":"ON",
"WRITE_BAS_HEAD":"ON",
"WRITE_BAS_VARIABLES":"ON",
"WRITE_DATA_COMPRESSED":0,
"WRITE_DATA_FORMAT":"DATA_FORMAT_EXTENSION",
"WRITE_DATA_PARAM":"OFF",
"WRITE_FREE_CON":"OFF",
"WRITE_GENERIC_NAMES":"OFF",
"WRITE_GENERIC_NAMES_IO":1,
"WRITE_IGNORE_INCOMPATIBLE_CONIC_ITEMS":"OFF",

```



```

    "WRITE_IGNORE_INCOMPATIBLE_ITEMS": "OFF",
    "WRITE_IGNORE_INCOMPATIBLE_NL_ITEMS": "OFF",
    "WRITE_IGNORE_INCOMPATIBLE_PSD_ITEMS": "OFF",
    "WRITE_INT_CONSTRAINTS": "ON",
    "WRITE_INT_HEAD": "ON",
    "WRITE_INT_VARIABLES": "ON",
    "WRITE_LP_FULL_OBJ": "ON",
    "WRITE_LP_LINE_WIDTH": 80,
    "WRITE_LP_QUOTED_NAMES": "ON",
    "WRITE_LP_STRICT_FORMAT": "OFF",
    "WRITE_LP_TERMS_PER_LINE": 10,
    "WRITE_MPS_FORMAT": "MPS_FORMAT_FREE",
    "WRITE_MPS_INT": "ON",
    "WRITE_PRECISION": 15,
    "WRITE_SOL_BARVARIABLES": "ON",
    "WRITE_SOL_CONSTRAINTS": "ON",
    "WRITE_SOL_HEAD": "ON",
    "WRITE_SOL_IGNORE_INVALID_NAMES": "OFF",
    "WRITE_SOL_VARIABLES": "ON",
    "WRITE_TASK_INC_SOL": "ON",
    "WRITE_XML_MODE": "WRITE_XML_MODE_ROW"
  },
  "dparam": {
    "ANA_SOL_INFEAS_TOL": 1e-6,
    "BASIS_REL_TOL_S": 1e-12,
    "BASIS_TOL_S": 1e-6,
    "BASIS_TOL_X": 1e-6,
    "CHECK_CONVEXITY_REL_TOL": 1e-10,
    "DATA_TOL_AIJ": 1e-12,
    "DATA_TOL_AIJ_HUGE": 1e+20,
    "DATA_TOL_AIJ_LARGE": 1e+10,
    "DATA_TOL_BOUND_INF": 1e+16,
    "DATA_TOL_BOUND_WRN": 1e+8,
    "DATA_TOL_C_HUGE": 1e+16,
    "DATA_TOL_CJ_LARGE": 1e+8,
    "DATA_TOL_QIJ": 1e-16,
    "DATA_TOL_X": 1e-8,
    "FEASREPAIR_TOL": 1e-10,
    "INTPNT_CO_TOL_DFEAS": 1e-8,
    "INTPNT_CO_TOL_INFEAS": 1e-10,
    "INTPNT_CO_TOL_MU_RED": 1e-8,
    "INTPNT_CO_TOL_NEAR_REL": 1e+3,
    "INTPNT_CO_TOL_PFEAS": 1e-8,
    "INTPNT_CO_TOL_REL_GAP": 1e-7,
    "INTPNT_NL_MERIT_BAL": 1e-4,
    "INTPNT_NL_TOL_DFEAS": 1e-8,
    "INTPNT_NL_TOL_MU_RED": 1e-12,
    "INTPNT_NL_TOL_NEAR_REL": 1e+3,
    "INTPNT_NL_TOL_PFEAS": 1e-8,
    "INTPNT_NL_TOL_REL_GAP": 1e-6,
    "INTPNT_NL_TOL_REL_STEP": 9.95e-1,
    "INTPNT_QO_TOL_DFEAS": 1e-8,
    "INTPNT_QO_TOL_INFEAS": 1e-10,
    "INTPNT_QO_TOL_MU_RED": 1e-8,
    "INTPNT_QO_TOL_NEAR_REL": 1e+3,
    "INTPNT_QO_TOL_PFEAS": 1e-8,
    "INTPNT_QO_TOL_REL_GAP": 1e-8,
    "INTPNT_TOL_DFEAS": 1e-8,
    "INTPNT_TOL_DSAFE": 1e+0,
    "INTPNT_TOL_INFEAS": 1e-10,
    "INTPNT_TOL_MU_RED": 1e-16,
    "INTPNT_TOL_PATH": 1e-8,
    "INTPNT_TOL_PFEAS": 1e-8,

```

```

    "INTPNT_TOL_PSAFE":1e+0,
    "INTPNT_TOL_REL_GAP":1e-8,
    "INTPNT_TOL_REL_STEP":9.999e-1,
    "INTPNT_TOL_STEP_SIZE":1e-6,
    "LOWER_OBJ_CUT":-1e+30,
    "LOWER_OBJ_CUT_FINITE_TRH":-5e+29,
    "MIO_DISABLE_TERM_TIME":-1e+0,
    "MIO_MAX_TIME":-1e+0,
    "MIO_MAX_TIME_APRX_OPT":6e+1,
    "MIO_NEAR_TOL_ABS_GAP":0.0,
    "MIO_NEAR_TOL_REL_GAP":1e-3,
    "MIO_REL_GAP_CONST":1e-10,
    "MIO_TOL_ABS_GAP":0.0,
    "MIO_TOL_ABS_RELAX_INT":1e-5,
    "MIO_TOL_FEAS":1e-6,
    "MIO_TOL_REL_DUAL_BOUND_IMPROVEMENT":0.0,
    "MIO_TOL_REL_GAP":1e-4,
    "MIO_TOL_X":1e-6,
    "OPTIMIZER_MAX_TIME":-1e+0,
    "PRESOLVE_TOL_ABS_LINDEP":1e-6,
    "PRESOLVE_TOL_AIJ":1e-12,
    "PRESOLVE_TOL_REL_LINDEP":1e-10,
    "PRESOLVE_TOL_S":1e-8,
    "PRESOLVE_TOL_X":1e-8,
    "QCQO_REFORMULATE_REL_DROP_TOL":1e-15,
    "SEMIDEFINITE_TOL_APPROX":1e-10,
    "SIM_LU_TOL_REL_PIV":1e-2,
    "SIMPLEX_ABS_TOL_PIV":1e-7,
    "UPPER_OBJ_CUT":1e+30,
    "UPPER_OBJ_CUT_FINITE_TRH":5e+29
  },
  "sparam":{
    "BAS_SOL_FILE_NAME":"",
    "DATA_FILE_NAME":"examples/tools/data/lo1.mps",
    "DEBUG_FILE_NAME":"",
    "INT_SOL_FILE_NAME":"",
    "ITR_SOL_FILE_NAME":"",
    "MIO_DEBUG_STRING":"",
    "PARAM_COMMENT_SIGN":"%%",
    "PARAM_READ_FILE_NAME":"",
    "PARAM_WRITE_FILE_NAME":"",
    "READ_MPS_BOU_NAME":"",
    "READ_MPS_OBJ_NAME":"",
    "READ_MPS_RAN_NAME":"",
    "READ_MPS_RHS_NAME":"",
    "SENSITIVITY_FILE_NAME":"",
    "SENSITIVITY_RES_FILE_NAME":"",
    "SOL_FILTER_XC_LOW":"",
    "SOL_FILTER_XC_UPR":"",
    "SOL_FILTER_XX_LOW":"",
    "SOL_FILTER_XX_UPR":"",
    "STAT_FILE_NAME":"",
    "STAT_KEY":"",
    "STAT_NAME":"",
    "WRITE_LP_GEN_VAR_NAME":"XMSKGEN"
  }
}

```

19.8 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
↔DUAL							CONIC
?	<name>	??	<a value>	<a value>	<a value>	<a value>	<a value>

In the example the fields ? and <> will be filled with problem and solution specific information. As can be observed a solution report consists of three sections, i.e.

- **HEADER** In this section, first the name of the problem is listed and afterwards the problem and solution status are shown. Next the primal and dual objective values are displayed.
- **CONSTRAINTS** For each constraint i of the form

$$l_i^c \leq \sum_{j=1}^n a_{ij}x_j \leq u_i^c, \quad (19.10)$$

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 19.4 the possible values of the status keys and their interpretation are shown.

Table 19.4: 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 (19.10).)
- **UPPER LIMIT**: the quantity u_i^c (see (19.10).)

- 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 19.7](#) we show the solution file for the `lo1.opf` problem.

Listing 19.7: 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 UPPER			
↪DUAL LOWER					
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 UPPER			
↪DUAL LOWER					
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			

LIST OF EXAMPLES

List of examples shipped in the distribution of Optimizer API for Java:

Table 20.1: List of distributed examples

File	Description
<code>blas_lapack.java</code>	Demonstrates the MOSEK interface to BLAS/LAPACK linear algebra routines
<code>callback.java</code>	An example of data/progress callback
<code>case_portfolio_1.java</code>	Implements a basic portfolio optimization model
<code>case_portfolio_2.java</code>	Implements a basic portfolio optimization model
<code>case_portfolio_3.java</code>	Implements a basic portfolio optimization model
<code>cqo1.java</code>	A simple conic quadratic problem
<code>feasreparex1.java</code>	A simple example of how to repair an infeasible problem
<code>lo1.java</code>	A simple linear problem
<code>lo2.java</code>	A simple linear problem
<code>milo1.java</code>	A simple mixed-integer linear problem
<code>miointsol.java</code>	A simple mixed-integer linear problem with an initial guess
<code>opt_server_async.java</code>	Uses MOSEK OptServer to solve an optimization problem asynchronously
<code>opt_server_sync.java</code>	Uses MOSEK OptServer to solve an optimization problem synchronously
<code>parameters.java</code>	Shows how to set optimizer parameters
<code>production.java</code>	Demonstrate how to modify and re-optimize a linear problem
<code>qcqo1.java</code>	A simple quadratically constrained quadratic problem
<code>qo1.java</code>	A simple quadratic problem
<code>response.java</code>	Demonstrates proper response handling
<code>scopt1.java</code>	Shows how to solve a simple non-linear separable problem using the SCoPt interface
<code>sdo1.java</code>	A simple semidefinite optimization problem
<code>sensitivity.java</code>	Sensitivity analysis performed on a small linear problem
<code>simple.java</code>	A simple I/O example: read problem from a file, solve and write solutions
<code>solutionquality.java</code>	Demonstrates how to examine the quality of a solution
<code>solvebasis.java</code>	Demonstrates solving a linear system with the basis matrix
<code>solvelinear.java</code>	Demonstrates solving a general linear system

Additional examples can be found on the **MOSEK** website and in other **MOSEK** publications.

INTERFACE CHANGES

The section show interface-specific changes to the **MOSEK** Optimizer API for Java in version 8. See the [release notes](#) for general changes and new features of the **MOSEK** Optimization Suite.

21.1 Compatibility

- Compatibility guarantees for this interface has been updated. See the new *state of compatibility*.

21.2 Functions

Added

Changed

Removed

- `Env.init`
- `Env.putdllpath`
- `Env.putkeepdlls`
- `Env.set_stream`
- `Task.getdbi`
- `Task.getdcni`
- `Task.getdeqi`
- `Task.getinti`
- `Task.getnumqconknz64`
- `Task.getpbi`
- `Task.getpcni`
- `Task.getpeqi`
- `Task.getqobj64`
- `Task.getsolutioninf`
- `Task.getvarbranchdir`
- `Task.getvarbranchorder`
- `Task.getvarbranchpri`

- `Task.progress`
- `Task.putvarbranchorder`
- `Task.readbranchpriorities`
- `Task.relaxprimal`
- `Task.set_stream`
- `Task.writebranchpriorities`

21.3 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`
- `iparam.remove_unused_solutions`
- `iparam.write_lp_full_obj`
- `iparam.write_mps_format`
- `sparam.remote_access_token`

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`
- `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_factor`
- `iparam.log_head`
- `iparam.log_nonconvex`
- `iparam.log_optimizer`
- `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`
- `iparam.read_cone`
- `iparam.read_mps_keep_int`
- `iparam.read_mps_obj_sense`
- `iparam.read_mps_relax`
- `iparam.read_qnz`
- `iparam.read_var`
- `iparam.sim_integer`
- `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`

21.4 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.qcgo_reformulate_max_perturbation*
- *dinfitem.qcgo_reformulate_worst_cholesky_column_scaling*
- *dinfitem.qcgo_reformulate_worst_cholesky_diag_scaling*
- *dinfitem.sol_bas_nrm_barx*
- *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_barx*
- *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*
- *solsta.dual_illposed_cer*
- *solsta.prim_illposed_cer*

Changed

- *solsta.integer_optimal*
- *solsta.near_dual_feas*
- *solsta.near_dual_infeas_cer*
- *solsta.near_integer_optimal*
- *solsta.near_optimal*
- *solsta.near_prim_and_dual_feas*
- *solsta.near_prim_feas*
- *solsta.near_prim_infeas_cer*
- *value.license_buffer_length*

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_primal_dual_simplex*
- *constant.callbackcode.begin_primal_dual_simplex_bi*
- *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_primal_dual_simplex`
- `constant.callbackcode.end_primal_dual_simplex_bi`
- `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.im_primal_dual_simplex`
- `constant.callbackcode.nonconvex`
- `constant.callbackcode.update_network_dual_simplex`
- `constant.callbackcode.update_network_primal_simplex`
- `constant.callbackcode.update_nonconvex`
- `constant.callbackcode.update_primal_dual_simplex`
- `constant.callbackcode.update_primal_dual_simplex_bi`
- `constant.dinfitem.bi_clean_primal_dual_time`
- `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.dinfitem.sim_primal_dual_time`
- `constant.feature.ptom`
- `constant.feature.ptox`
- `constant.iinfitem.concurrent_fastest_optimizer`
- `constant.iinfitem.mio_num_basis_cuts`
- `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.sim_primal_dual_deg_iter`
- `constant.iinfitem.sim_primal_dual_hotstart`
- `constant.iinfitem.sim_primal_dual_hotstart_lu`
- `constant.iinfitem.sim_primal_dual_inf_iter`
- `constant.iinfitem.sim_primal_dual_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.liinfitem.bi_clean_primal_dual_deg_iter`
- `constant.liinfitem.bi_clean_primal_dual_iter`
- `constant.liinfitem.bi_clean_primal_dual_sub_iter`
- `constant.miomode.lazy`
- `constant.optimizertype.concurrent`
- `constant.optimizertype.mixed_int_conic`
- `constant.optimizertype.network_primal_simplex`
- `constant.optimizertype.nonconvex`
- `constant.optimizertype.primal_dual_simplex`

21.5 Response Codes

Added

- *`rescode.err_cbf_duplicate_psdvar`*
- *`rescode.err_cbf_invalid_psdvar_dimension`*
- *`rescode.err_cbf_too_few_psdvar`*
- *`rescode.err_duplicate_aij`*
- *`rescode.err_final_solution`*

- *rescode.err_json_data*
- *rescode.err_json_format*
- *rescode.err_json_missing_data*
- *rescode.err_json_number_overflow*
- *rescode.err_json_string*
- *rescode.err_json_syntax*
- *rescode.err_lau_invalid_lower_triangular_matrix*
- *rescode.err_lau_invalid_sparse_symmetric_matrix*
- *rescode.err_lau_not_positive_definite*
- *rescode.err_mixed_conic_and_nl*
- *rescode.err_server_connect*
- *rescode.err_server_protocol*
- *rescode.err_server_status*
- *rescode.err_server_token*
- *rescode.err_sym_mat_huge*
- *rescode.err_sym_mat_invalid*
- *rescode.err_task_write*
- *rescode.err_toconic_constr_not_conic*
- *rescode.err_toconic_constr_q_not_psd*
- *rescode.err_toconic_constraint_fx*
- *rescode.err_toconic_constraint_ra*
- *rescode.err_toconic_objective_not_psd*
- *rescode.wrn_sym_mat_large*

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_mio_not_loaded*
- *rescode.err_mixed_problem*
- *rescode.err_no_dual_info_for_itg_sol*

- `rescode.err_ord_invalid`
- `rescode.err_ord_invalid_branch_dir`
- `rescode.err_toconic_conversion_fail`
- `rescode.err_too_many_concurrent_tasks`
- `rescode.wrn_too_many_threads_concurrent`

- [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>.
- [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.
- [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>.

Classes

DataCallback, 376
 DataCallback.callback, 376
 Env, 174
 Env.syrk, 183
 Env.syevd, 182
 Env.syeig, 182
 Env.sparsetriangularsolvedense, 181
 Env.set_Stream, 181
 Env.putlicensewait, 181
 Env.putlicensepath, 181
 Env.putlicensedebug, 180
 Env.putlicensecode, 180
 Env.potrf, 180
 Env.linkfiletostream, 180
 Env.licensecleanup, 179
 Env.getversion, 179
 Env.getcodedesc, 179
 Env.gemv, 178
 Env.gemm, 177
 Env.Env, 174
 Env.echointro, 177
 Env.dot, 177
 Env.computesparsecholesky, 175
 Env.checkoutlicense, 175
 Env.checkinlicense, 175
 Env.checkinall, 175
 Env.axy, 174
 ItgSolutionCallback, 375
 ItgSolutionCallback.callback, 375
 Progress, 375
 Progress.progress, 375
 Stream, 376
 Stream.stream, 376
 Task, 184
 Task.writetask, 279
 Task.writesolution, 279
 Task.writeSC, 378
 Task.writeparamfile, 279
 Task.writejsonsol, 278
 Task.writedata, 278
 Task.updatesolutioninfo, 278
 Task.unset_Progress, 278
 Task.toconic, 277
 Task.Task, 184
 Task.strtosk, 277

Task.strtoconetype, 277
 Task.solvewithbasis, 276
 Task.solutionsummary, 276
 Task.solutiondef, 275
 Task.setdefaults, 275
 Task.set_Stream, 275
 Task.set_Progress, 275
 Task.set_ItgSolutionCallback, 274
 Task.set_InfoCallback, 274
 Task.sensitivityreport, 274
 Task.resizetask, 274
 Task.removevars, 273
 Task.removecons, 273
 Task.removecones, 273
 Task.removebarvars, 273
 Task.readtask, 273
 Task.readsummary, 272
 Task.readsolution, 272
 Task.readparamfile, 272
 Task.readdataformat, 272
 Task.readdata, 271
 Task.putyslice, 271
 Task.puty, 271
 Task.putxxslice, 271
 Task.putxx, 270
 Task.putxcslice, 270
 Task.putxc, 270
 Task.putvartypelist, 269
 Task.putvartype, 269
 Task.putvarname, 269
 Task.putvarboundslice, 269
 Task.putvarboundlist, 268
 Task.putvarbound, 268
 Task.puttaskname, 268
 Task.putsukslice, 267
 Task.putsuks, 267
 Task.putsucslice, 267
 Task.putsuc, 267
 Task.putstrparam, 266
 Task.putsolutionyi, 266
 Task.putsolutioni, 265
 Task.putsolution, 265
 Task.putsnxslice, 264
 Task.putsnx, 264
 Task.putslxslice, 264
 Task.putslx, 263
 Task.putslcslice, 263

Task.putslc, 263
Task.putskxslice, 263
Task.putskx, 262
Task.putskcslice, 262
Task.putskc, 262
Task.putSCeval, 377
Task.putqobjij, 261
Task.putqobj, 261
Task.putqconk, 260
Task.putqcon, 260
Task.putparam, 260
Task.putobjsense, 259
Task.putobjname, 259
Task.putnastrparam, 259
Task.putnaintparam, 259
Task.putnadoupparam, 258
Task.putmaxnumvar, 258
Task.putmaxnumqnz, 258
Task.putmaxnumcone, 258
Task.putmaxnumcon, 257
Task.putmaxnumbarvar, 257
Task.putmaxnumanz, 257
Task.putintparam, 256
Task.putdoupparam, 256
Task.putcslice, 256
Task.putconname, 255
Task.putconename, 255
Task.putcone, 255
Task.putconboundslice, 254
Task.putconboundlist, 254
Task.putconbound, 254
Task.putclist, 253
Task.putcj, 253
Task.putcfix, 253
Task.putboundslice, 252
Task.putboundlist, 252
Task.putbound, 252
Task.putbarxj, 251
Task.putbarvarname, 251
Task.putbarsj, 251
Task.putbarcj, 250
Task.putbarcblocktriplet, 250
Task.putbaraij, 249
Task.putbarablocktriplet, 249
Task.putarowslice, 248
Task.putarowlist, 248
Task.putarow, 248
Task.putaijlist, 247
Task.putaij, 247
Task.putacolslice, 246
Task.putacollist, 246
Task.putacol, 245
Task.primalsensitivity, 244
Task.primalrepair, 244
Task.optimizersummary, 243
Task.optimizermt, 243
Task.optimize, 243
Task.onesolutionsummary, 243
Task.linkfiletostream, 242
Task.isstrparname, 242
Task.isintparname, 242
Task.isdoupname, 241
Task.inputdata, 240
Task.initbasissolve, 240
Task.getyslice, 240
Task.gety, 239
Task.getxxslice, 239
Task.getxx, 239
Task.getxcslice, 239
Task.getxc, 238
Task.getvartypelist, 238
Task.getvartype, 238
Task.getvarnamelen, 237
Task.getvarnameindex, 237
Task.getvarname, 237
Task.getvarboundslice, 236
Task.getvarbound, 236
Task.gettasknamelen, 236
Task.gettaskname, 235
Task.getsymmatinfo, 235
Task.getsuxslice, 235
Task.getsux, 234
Task.getsucslice, 234
Task.getsuc, 234
Task.getstrparamlen, 234
Task.getstrparam, 233
Task.getsparsesymmat, 233
Task.getsolutionslice, 232
Task.getsolutioninfo, 231
Task.getsolutioni, 231
Task.getsolution, 229
Task.getsolsta, 229
Task.getsnxslice, 228
Task.getsnx, 228
Task.getslxslice, 228
Task.getslx, 228
Task.getslcslice, 227
Task.getslc, 227
Task.getskxslice, 227
Task.getskx, 226
Task.getskcslice, 226
Task.getskc, 226
Task.getreducedcosts, 226
Task.getqobjij, 225
Task.getqobj, 225
Task.getqconk, 224
Task.getpviolvar, 224
Task.getpviolcones, 223
Task.getpviolcon, 223
Task.getpviolbarvar, 223
Task.getprosta, 222
Task.getprobtype, 222
Task.getprimalsolutionnorms, 222
Task.getprimalobj, 221
Task.getparamname, 221
Task.getparammax, 221

Task.getobjsense, 221
 Task.getobjnamelen, 220
 Task.getobjname, 220
 Task.getnumvar, 220
 Task.getnumsymmat, 220
 Task.getnumqobjnz, 219
 Task.getnumqconknz, 219
 Task.getnumparam, 219
 Task.getnumintvar, 219
 Task.getnumconemem, 218
 Task.getnumcone, 218
 Task.getnumcon, 218
 Task.getnumbarvar, 218
 Task.getnumbarcnz, 217
 Task.getnumbarcbloctriplets, 217
 Task.getnumbaranz, 217
 Task.getnumbarablocktriplets, 217
 Task.getnumanz64, 217
 Task.getnumanz, 216
 Task.getmemusage, 216
 Task.getmaxnumvar, 216
 Task.getmaxnumqnz, 216
 Task.getmaxnumcone, 215
 Task.getmaxnumcon, 215
 Task.getmaxnumbarvar, 215
 Task.getmaxnumanz, 215
 Task.getlintinf, 214
 Task.getlenbarvarj, 214
 Task.getintparam, 214
 Task.getintinf, 213
 Task.getinfname, 213
 Task.getinfmax, 213
 Task.getinfindex, 213
 Task.getinfeasiblesubproblem, 212
 Task.getdviolvar, 212
 Task.getdviolcones, 211
 Task.getdviolcon, 211
 Task.getdviolbarvar, 210
 Task.getdualsolutionnorms, 210
 Task.getdualobj, 210
 Task.getdoupparam, 209
 Task.getdouinf, 209
 Task.getdimbarvarj, 209
 Task.getcslice, 208
 Task.getconnamelen, 208
 Task.getconnameindex, 208
 Task.getconname, 207
 Task.getconenamelen, 207
 Task.getconenameindex, 206
 Task.getconename, 206
 Task.getconeinfo, 206
 Task.getcone, 205
 Task.getconboundslice, 205
 Task.getconbound, 205
 Task.getcj, 204
 Task.getcfix, 204
 Task.getc, 204
 Task.getboundslice, 204
 Task.getbound, 203
 Task.getbarxj, 203
 Task.getbarvarnamelen, 203
 Task.getbarvarnameindex, 202
 Task.getbarvarname, 202
 Task.getbarsj, 202
 Task.getbarcsparsity, 201
 Task.getbarcidxj, 201
 Task.getbarcidxinfo, 201
 Task.getbarcidx, 200
 Task.getbarcbloctriplet, 200
 Task.getbarasparsity, 199
 Task.getbaraidxinfo, 199
 Task.getbaraidxij, 199
 Task.getbaraidx, 198
 Task.getbarablocktriplet, 197
 Task.getaslicenumnz, 197
 Task.getaslice, 196
 Task.getarowslicetrip, 196
 Task.getarownumnz, 195
 Task.getarow, 195
 Task.getapiecenumnz, 195
 Task.getaij, 194
 Task.getacolslicetrip, 194
 Task.getacolnumnz, 194
 Task.getacol, 193
 Task.dualsensitivity, 193
 Task.deletesolution, 192
 Task.commitchanges, 192
 Task.clearSCeval, 378
 Task.chgvarbound, 192
 Task.chgconbound, 191
 Task.chgbound, 191
 Task.checkmem, 190
 Task.checkconvexity, 190
 Task.basiscond, 190
 Task.asyncstop, 190
 Task.asyncpoll, 189
 Task.asyncoptimize, 189
 Task.asyncgetresult, 188
 Task.appendvars, 188
 Task.appendsparsesymmat, 187
 Task.appendcons, 187
 Task.appendconesseq, 186
 Task.appendconeseq, 186
 Task.appendcone, 185
 Task.appendbarvars, 185
 Task.analyzesolution, 185
 Task.analyzeproblem, 185
 Task.analyzenames, 184

Enumerations

accmode, 350
 accmode.var, 350
 accmode.con, 350
 basindtype, 350
 basindtype.reserved, 350
 basindtype.no_error, 350

basindtype.never, 350
 basindtype.if_feasible, 350
 basindtype.always, 350
 boundkey, 350
 boundkey.up, 350
 boundkey.ra, 351
 boundkey.lo, 350
 boundkey.fx, 350
 boundkey.fr, 351
 branchdir, 367
 branchdir.up, 367
 branchdir.root_lp, 368
 branchdir.pseudocost, 368
 branchdir.near, 367
 branchdir.guided, 368
 branchdir.free, 367
 branchdir.far, 367
 branchdir.down, 367
 callbackcode, 352
 callbackcode.write_opf, 357
 callbackcode.update_primal_simplex_bi, 357
 callbackcode.update_primal_simplex, 356
 callbackcode.update_primal_bi, 356
 callbackcode.update_presolve, 356
 callbackcode.update_dual_simplex_bi, 356
 callbackcode.update_dual_simplex, 356
 callbackcode.update_dual_bi, 356
 callbackcode.solving_remote, 356
 callbackcode.read_opf_section, 356
 callbackcode.read_opf, 356
 callbackcode.primal_simplex, 356
 callbackcode.new_int_mio, 356
 callbackcode.intpnt, 356
 callbackcode.im_simplex_bi, 356
 callbackcode.im_simplex, 356
 callbackcode.im_root_cutgen, 356
 callbackcode.im_read, 356
 callbackcode.im_qo_reformulate, 356
 callbackcode.im_primal_simplex, 356
 callbackcode.im_primal_sensitivity, 356
 callbackcode.im_primal_bi, 355
 callbackcode.im_presolve, 355
 callbackcode.im_order, 355
 callbackcode.im_mio_primal_simplex, 355
 callbackcode.im_mio_intpnt, 355
 callbackcode.im_mio_dual_simplex, 355
 callbackcode.im_mio, 355
 callbackcode.im_lu, 355
 callbackcode.im_license_wait, 355
 callbackcode.im_intpnt, 355
 callbackcode.im_full_convexity_check, 355
 callbackcode.im_dual_simplex, 355
 callbackcode.im_dual_sensitivity, 355
 callbackcode.im_dual_bi, 355
 callbackcode.im_conic, 355
 callbackcode.im_bi, 355
 callbackcode.end_write, 355
 callbackcode.end_to_conic, 355
 callbackcode.end_simplex_bi, 355
 callbackcode.end_simplex, 355
 callbackcode.end_root_cutgen, 354
 callbackcode.end_read, 354
 callbackcode.end_qcgo_reformulate, 354
 callbackcode.end_primal_simplex_bi, 354
 callbackcode.end_primal_simplex, 354
 callbackcode.end_primal_setup_bi, 354
 callbackcode.end_primal_sensitivity, 354
 callbackcode.end_primal_repair, 354
 callbackcode.end_primal_bi, 354
 callbackcode.end_presolve, 354
 callbackcode.end_optimizer, 354
 callbackcode.end_mio, 354
 callbackcode.end_license_wait, 354
 callbackcode.end_intpnt, 354
 callbackcode.end_infeas_ana, 354
 callbackcode.end_full_convexity_check, 354
 callbackcode.end_dual_simplex_bi, 354
 callbackcode.end_dual_simplex, 354
 callbackcode.end_dual_setup_bi, 354
 callbackcode.end_dual_sensitivity, 354
 callbackcode.end_dual_bi, 354
 callbackcode.end_conic, 354
 callbackcode.end_bi, 353
 callbackcode.dual_simplex, 353
 callbackcode.conic, 353
 callbackcode.begin_write, 353
 callbackcode.begin_to_conic, 353
 callbackcode.begin_simplex_bi, 353
 callbackcode.begin_simplex, 353
 callbackcode.begin_root_cutgen, 353
 callbackcode.begin_read, 353
 callbackcode.begin_qcgo_reformulate, 353
 callbackcode.begin_primal_simplex_bi, 353
 callbackcode.begin_primal_simplex, 353
 callbackcode.begin_primal_setup_bi, 353
 callbackcode.begin_primal_sensitivity, 353
 callbackcode.begin_primal_repair, 353
 callbackcode.begin_primal_bi, 353
 callbackcode.begin_presolve, 353
 callbackcode.begin_optimizer, 353
 callbackcode.begin_mio, 353
 callbackcode.begin_license_wait, 353
 callbackcode.begin_intpnt, 353
 callbackcode.begin_infeas_ana, 352
 callbackcode.begin_full_convexity_check, 352
 callbackcode.begin_dual_simplex_bi, 352
 callbackcode.begin_dual_simplex, 352
 callbackcode.begin_dual_setup_bi, 352
 callbackcode.begin_dual_sensitivity, 352
 callbackcode.begin_dual_bi, 352
 callbackcode.begin_conic, 352
 callbackcode.begin_bi, 352
 checkconvexitytype, 357
 checkconvexitytype.simple, 357
 checkconvexitytype.none, 357

checkconvexitytype.full, 357
 compresstype, 357
 compresstype.none, 357
 compresstype.gzip, 357
 compresstype.free, 357
 conetype, 357
 conetype.rquad, 357
 conetype.quad, 357
 dataformat, 357
 dataformat.xml, 358
 dataformat.task, 358
 dataformat.op, 358
 dataformat.mps, 357
 dataformat.lp, 357
 dataformat.json_task, 358
 dataformat.free_mps, 358
 dataformat.extension, 357
 dataformat.cb, 358
 dinfitem, 358
 dinfitem.to_conic_time, 362
 dinfitem.sol_itr_pviolvar, 362
 dinfitem.sol_itr_pviolcones, 362
 dinfitem.sol_itr_pviolcon, 362
 dinfitem.sol_itr_pviolbarvar, 362
 dinfitem.sol_itr_primal_obj, 362
 dinfitem.sol_itr_nrm_y, 362
 dinfitem.sol_itr_nrm_xx, 362
 dinfitem.sol_itr_nrm_xc, 362
 dinfitem.sol_itr_nrm_sux, 362
 dinfitem.sol_itr_nrm_suc, 362
 dinfitem.sol_itr_nrm_snx, 362
 dinfitem.sol_itr_nrm_slx, 362
 dinfitem.sol_itr_nrm_slc, 362
 dinfitem.sol_itr_nrm_barx, 362
 dinfitem.sol_itr_nrm_bars, 362
 dinfitem.sol_itr_dviolvar, 361
 dinfitem.sol_itr_dviolcones, 361
 dinfitem.sol_itr_dviolcon, 361
 dinfitem.sol_itr_dviolbarvar, 361
 dinfitem.sol_itr_dual_obj, 361
 dinfitem.sol_itg_pviolvar, 361
 dinfitem.sol_itg_pviolitg, 361
 dinfitem.sol_itg_pviolcones, 361
 dinfitem.sol_itg_pviolcon, 361
 dinfitem.sol_itg_pviolbarvar, 361
 dinfitem.sol_itg_primal_obj, 361
 dinfitem.sol_itg_nrm_xx, 361
 dinfitem.sol_itg_nrm_xc, 361
 dinfitem.sol_itg_nrm_barx, 361
 dinfitem.sol_bas_pviolvar, 361
 dinfitem.sol_bas_pviolcon, 361
 dinfitem.sol_bas_primal_obj, 361
 dinfitem.sol_bas_nrm_y, 361
 dinfitem.sol_bas_nrm_xx, 361
 dinfitem.sol_bas_nrm_xc, 361
 dinfitem.sol_bas_nrm_sux, 361
 dinfitem.sol_bas_nrm_suc, 361
 dinfitem.sol_bas_nrm_slx, 361
 dinfitem.sol_bas_nrm_slc, 360
 dinfitem.sol_bas_nrm_barx, 360
 dinfitem.sol_bas_dviolvar, 360
 dinfitem.sol_bas_dviolcon, 360
 dinfitem.sol_bas_dual_obj, 360
 dinfitem.sim_time, 360
 dinfitem.sim_primal_time, 360
 dinfitem.sim_obj, 360
 dinfitem.sim_feas, 360
 dinfitem.sim_dual_time, 360
 dinfitem.rd_time, 360
 dinfitem.qcqp_reformulate_worst_cholesky_diag_scaling, 360
 dinfitem.qcqp_reformulate_worst_cholesky_column_scaling, 360
 dinfitem.qcqp_reformulate_time, 360
 dinfitem.qcqp_reformulate_max_perturbation, 360
 dinfitem.primal_repair_penalty_obj, 360
 dinfitem.presolve_time, 360
 dinfitem.presolve_lindep_time, 360
 dinfitem.presolve_eli_time, 360
 dinfitem.optimizer_time, 360
 dinfitem.mio_user_obj_cut, 360
 dinfitem.mio_time, 360
 dinfitem.mio_root_presolve_time, 360
 dinfitem.mio_root_optimizer_time, 359
 dinfitem.mio_root_cutgen_time, 359
 dinfitem.mio_probing_time, 359
 dinfitem.mio_optimizer_time, 359
 dinfitem.mio_obj_rel_gap, 359
 dinfitem.mio_obj_int, 359
 dinfitem.mio_obj_bound, 359
 dinfitem.mio_obj_abs_gap, 359
 dinfitem.mio_knapsack_cover_separation_time, 359
 dinfitem.mio_implied_bound_time, 359
 dinfitem.mio_heuristic_time, 359
 dinfitem.mio_gmi_separation_time, 359
 dinfitem.mio_dual_bound_after_presolve, 359
 dinfitem.mio_construct_solution_obj, 359
 dinfitem.mio_cmir_separation_time, 359
 dinfitem.mio_clique_separation_time, 359
 dinfitem.intpnt_time, 359
 dinfitem.intpnt_primal_obj, 358
 dinfitem.intpnt_primal_feas, 358
 dinfitem.intpnt_order_time, 358
 dinfitem.intpnt_opt_status, 358
 dinfitem.intpnt_factor_num_flops, 358
 dinfitem.intpnt_dual_obj, 358
 dinfitem.intpnt_dual_feas, 358
 dinfitem.bi_time, 358
 dinfitem.bi_primal_time, 358
 dinfitem.bi_dual_time, 358
 dinfitem.bi_clean_time, 358
 dinfitem.bi_clean_primal_time, 358
 dinfitem.bi_clean_dual_time, 358
 dparam, 291

feature, 362
feature.pts, 362
feature.pton, 362
iinfitem, 363
iinfitem.sto_num_a_realloc, 367
iinfitem.sol_itr_solsta, 367
iinfitem.sol_itr_prosta, 367
iinfitem.sol_itg_solsta, 367
iinfitem.sol_itg_prosta, 367
iinfitem.sol_bas_solsta, 367
iinfitem.sol_bas_prosta, 367
iinfitem.sim_solve_dual, 367
iinfitem.sim_primal_iter, 366
iinfitem.sim_primal_inf_iter, 366
iinfitem.sim_primal_hotstart_lu, 366
iinfitem.sim_primal_hotstart, 366
iinfitem.sim_primal_deg_iter, 366
iinfitem.sim_numvar, 366
iinfitem.sim_numcon, 366
iinfitem.sim_dual_iter, 366
iinfitem.sim_dual_inf_iter, 366
iinfitem.sim_dual_hotstart_lu, 366
iinfitem.sim_dual_hotstart, 366
iinfitem.sim_dual_deg_iter, 366
iinfitem.rd_prototype, 366
iinfitem.rd_numvar, 366
iinfitem.rd_numq, 366
iinfitem.rd_numintvar, 366
iinfitem.rd_numcone, 366
iinfitem.rd_numcon, 366
iinfitem.rd_numbarvar, 366
iinfitem.optimize_response, 366
iinfitem.opt_numvar, 366
iinfitem.opt_numcon, 366
iinfitem.mio_user_obj_cut, 365
iinfitem.mio_total_num_cuts, 365
iinfitem.mio_relgap_satisfied, 365
iinfitem.mio_presolved_numvar, 365
iinfitem.mio_presolved_numint, 365
iinfitem.mio_presolved_numcont, 365
iinfitem.mio_presolved_numcon, 365
iinfitem.mio_presolved_numbin, 365
iinfitem.mio_obj_bound_defined, 365
iinfitem.mio_numvar, 365
iinfitem.mio_numint, 365
iinfitem.mio_numcon, 365
iinfitem.mio_num_repeated_presolve, 365
iinfitem.mio_num_relax, 365
iinfitem.mio_num_knapsack_cover_cuts, 365
iinfitem.mio_num_int_solutions, 365
iinfitem.mio_num_implied_bound_cuts, 365
iinfitem.mio_num_gomory_cuts, 365
iinfitem.mio_num_cmir_cuts, 365
iinfitem.mio_num_clique_cuts, 365
iinfitem.mio_num_branch, 365
iinfitem.mio_num_active_nodes, 365
iinfitem.mio_node_depth, 365
iinfitem.mio_near_relgap_satisfied, 365
iinfitem.mio_near_absgap_satisfied, 364
iinfitem.mio_initial_solution, 364
iinfitem.mio_construct_solution, 364
iinfitem.mio_construct_num_roundings, 364
iinfitem.mio_clique_table_size, 364
iinfitem.mio_absgap_satisfied, 364
iinfitem.intpnt_solve_dual, 364
iinfitem.intpnt_num_threads, 364
iinfitem.intpnt_iter, 364
iinfitem.intpnt_factor_dim_dense, 364
iinfitem.ana_pro_num_var_up, 364
iinfitem.ana_pro_num_var_ra, 364
iinfitem.ana_pro_num_var_lo, 364
iinfitem.ana_pro_num_var_int, 364
iinfitem.ana_pro_num_var_fr, 364
iinfitem.ana_pro_num_var_eq, 364
iinfitem.ana_pro_num_var_cont, 364
iinfitem.ana_pro_num_var_bin, 364
iinfitem.ana_pro_num_var, 363
iinfitem.ana_pro_num_con_up, 363
iinfitem.ana_pro_num_con_ra, 363
iinfitem.ana_pro_num_con_lo, 363
iinfitem.ana_pro_num_con_fr, 363
iinfitem.ana_pro_num_con_eq, 363
iinfitem.ana_pro_num_con, 363
infetype, 367
infetype.lint_type, 367
infetype.int_type, 367
infetype.dou_type, 367
intpnthotstart, 352
intpnthotstart.primal_dual, 352
intpnthotstart.primal, 352
intpnthotstart.none, 352
intpnthotstart.dual, 352
iomode, 367
iomode.write, 367
iomode.readwrite, 367
iomode.read, 367
iparam, 301
language, 350
language.eng, 350
language.dan, 350
liinfitem, 362
liinfitem.rd_numqnz, 363
liinfitem.rd_numanz, 363
liinfitem.mio_simplex_iter, 363
liinfitem.mio_sim_maxiter_setbacks, 363
liinfitem.mio_presolved_anz, 363
liinfitem.mio_intpnt_iter, 363
liinfitem.intpnt_factor_num_nz, 363
liinfitem.bi_primal_iter, 363
liinfitem.bi_dual_iter, 363
liinfitem.bi_clean_primal_iter, 363
liinfitem.bi_clean_primal_deg_iter, 362
liinfitem.bi_clean_dual_iter, 362
liinfitem.bi_clean_dual_deg_iter, 362
mark, 351
mark.up, 351

mark.lo, 351
 miocontsoltype, 368
 miocontsoltype.root, 368
 miocontsoltype.none, 368
 miocontsoltype.itg_rel, 368
 miocontsoltype.itg, 368
 miomode, 368
 miomode.satisfied, 368
 miomode.ignored, 368
 mionodeseltype, 368
 mionodeseltype.worst, 368
 mionodeseltype.pseudo, 368
 mionodeseltype.hybrid, 368
 mionodeseltype.free, 368
 mionodeseltype.first, 368
 mionodeseltype.best, 368
 mpsformat, 368
 mpsformat.strict, 368
 mpsformat.relaxed, 368
 mpsformat.free, 369
 mpsformat.cplex, 369
 nametype, 357
 nametype.mps, 357
 nametype.lp, 357
 nametype.gen, 357
 objsense, 369
 objsense.minimize, 369
 objsense.maximize, 369
 onoffkey, 369
 onoffkey.on, 369
 onoffkey.off, 369
 optimizertype, 369
 optimizertype.primal_simplex, 369
 optimizertype.mixed_int, 369
 optimizertype.intpnt, 369
 optimizertype.free_simplex, 369
 optimizertype.free, 369
 optimizertype.dual_simplex, 369
 optimizertype.conic, 369
 orderingtype, 369
 orderingtype.try_graphpar, 369
 orderingtype.none, 370
 orderingtype.free, 369
 orderingtype.force_graphpar, 369
 orderingtype.experimental, 369
 orderingtype.appminloc, 369
 parametertype, 370
 parametertype.str_type, 370
 parametertype.invalid_type, 370
 parametertype.int_type, 370
 parametertype.dou_type, 370
 presolvemode, 370
 presolvemode.on, 370
 presolvemode.off, 370
 presolvemode.free, 370
 problemitem, 370
 problemitem.var, 370
 problemitem.cone, 370
 problemitem.con, 370
 problemtype, 370
 problemtype.qo, 370
 problemtype.qcqp, 370
 problemtype.mixed, 370
 problemtype.lo, 370
 problemtype.geco, 370
 problemtype.conic, 370
 prosta, 370
 prosta.unknown, 371
 prosta.prim_infeas_or_unbounded, 371
 prosta.prim_infeas, 371
 prosta.prim_feas, 371
 prosta.prim_and_dual_infeas, 371
 prosta.prim_and_dual_feas, 371
 prosta.near_prim_feas, 371
 prosta.near_prim_and_dual_feas, 371
 prosta.near_dual_feas, 371
 prosta.ill_posed, 371
 prosta.dual_infeas, 371
 prosta.dual_feas, 371
 rescode, 328
 rescodetype, 371
 rescodetype.wrn, 371
 rescodetype.unk, 371
 rescodetype.trm, 371
 rescodetype.ok, 371
 rescodetype.err, 371
 scalingmethod, 372
 scalingmethod.pow2, 372
 scalingmethod.free, 372
 scalingtype, 371
 scalingtype.none, 372
 scalingtype.moderate, 372
 scalingtype.free, 372
 scalingtype.aggressive, 372
 scopr, 377
 sensitivitytype, 372
 sensitivitytype.optimal_partition, 372
 sensitivitytype.basis, 372
 simdegen, 351
 simdegen.none, 351
 simdegen.moderate, 351
 simdegen.minimum, 351
 simdegen.free, 351
 simdegen.aggressive, 351
 simdupvec, 351
 simdupvec.on, 352
 simdupvec.off, 352
 simdupvec.free, 352
 simhotstart, 352
 simhotstart.status_keys, 352
 simhotstart.none, 352
 simhotstart.free, 352
 simreform, 351
 simreform.on, 351
 simreform.off, 351
 simreform.free, 351

- simreform.aggressive, 351
- simseltype, 372
- simseltype.se, 372
- simseltype.partial, 372
- simseltype.full, 372
- simseltype.free, 372
- simseltype.devex, 372
- simseltype.ase, 372
- solitem, 372
- solitem.y, 372
- solitem.xx, 372
- solitem.xc, 372
- solitem.sux, 373
- solitem.suc, 373
- solitem.snx, 373
- solitem.slx, 373
- solitem.slc, 372
- solsta, 373
- solsta.unknown, 373
- solsta.prim_infeas_cer, 373
- solsta.prim_illposed_cer, 373
- solsta.prim_feas, 373
- solsta.prim_and_dual_feas, 373
- solsta.optimal, 373
- solsta.near_prim_infeas_cer, 373
- solsta.near_prim_feas, 373
- solsta.near_prim_and_dual_feas, 373
- solsta.near_optimal, 373
- solsta.near_integer_optimal, 373
- solsta.near_dual_infeas_cer, 373
- solsta.near_dual_feas, 373
- solsta.integer_optimal, 373
- solsta.dual_infeas_cer, 373
- solsta.dual_illposed_cer, 373
- solsta.dual_feas, 373
- soltype, 373
- soltype.itr, 374
- soltype.itg, 374
- soltype.bas, 374
- solveform, 374
- solveform.primal, 374
- solveform.free, 374
- solveform.dual, 374
- sparam, 326
- stakey, 374
- stakey.upr, 374
- stakey.unk, 374
- stakey.supbas, 374
- stakey.low, 374
- stakey.inf, 374
- stakey.fix, 374
- stakey.bas, 374
- startpointtype, 374
- startpointtype.satisfy_bounds, 374
- startpointtype.guess, 374
- startpointtype.free, 374
- startpointtype.constant, 374
- streamtype, 374

- streamtype.wrn, 375
- streamtype.msg, 375
- streamtype.log, 374
- streamtype.err, 375
- symmattype, 357
- symmattype.sparse, 357
- transpose, 351
- transpose.yes, 351
- transpose.no, 351
- uplo, 351
- uplo.up, 351
- uplo.lo, 351
- value, 375
- value.max_str_len, 375
- value.license_buffer_length, 375
- variabletype, 375
- variabletype.type_int, 375
- variabletype.type_cont, 375
- xmlwriteroutputtype, 371
- xmlwriteroutputtype.row, 371
- xmlwriteroutputtype.col, 371

Exceptions

- ArrayLengthException, 280
- Error, 279
- Exception, 279
- MosekException, 279
- NullPointerException, 279
- Warning, 279

Parameters

- Double parameters, 291
- dparam.ana_sol_infeas_tol, 291
- dparam.basis_rel_tol_s, 291
- dparam.basis_tol_s, 291
- dparam.basis_tol_x, 291
- dparam.check_convexity_rel_tol, 292
- dparam.data_sym_mat_tol, 292
- dparam.data_sym_mat_tol_huge, 292
- dparam.data_sym_mat_tol_large, 292
- dparam.data_tol_aij, 292
- dparam.data_tol_aij_huge, 292
- dparam.data_tol_aij_large, 293
- dparam.data_tol_bound_inf, 293
- dparam.data_tol_bound_wrn, 293
- dparam.data_tol_c_huge, 293
- dparam.data_tol_cj_large, 293
- dparam.data_tol_qij, 293
- dparam.data_tol_x, 293
- dparam.intpnt_co_tol_dfeas, 293
- dparam.intpnt_co_tol_infeas, 294
- dparam.intpnt_co_tol_mu_red, 294
- dparam.intpnt_co_tol_near_rel, 294
- dparam.intpnt_co_tol_pfeas, 294
- dparam.intpnt_co_tol_rel_gap, 294
- dparam.intpnt_nl_merit_bal, 294
- dparam.intpnt_nl_tol_dfeas, 294
- dparam.intpnt_nl_tol_mu_red, 295

dparam.intpnt_nl_tol_near_rel, 295
 dparam.intpnt_nl_tol_pfeas, 295
 dparam.intpnt_nl_tol_rel_gap, 295
 dparam.intpnt_nl_tol_rel_step, 295
 dparam.intpnt_qo_tol_dfeas, 295
 dparam.intpnt_qo_tol_infeas, 295
 dparam.intpnt_qo_tol_mu_red, 295
 dparam.intpnt_qo_tol_near_rel, 296
 dparam.intpnt_qo_tol_pfeas, 296
 dparam.intpnt_qo_tol_rel_gap, 296
 dparam.intpnt_tol_dfeas, 296
 dparam.intpnt_tol_dsafe, 296
 dparam.intpnt_tol_infeas, 296
 dparam.intpnt_tol_mu_red, 297
 dparam.intpnt_tol_path, 297
 dparam.intpnt_tol_pfeas, 297
 dparam.intpnt_tol_psafe, 297
 dparam.intpnt_tol_rel_gap, 297
 dparam.intpnt_tol_rel_step, 297
 dparam.intpnt_tol_step_size, 297
 dparam.lower_obj_cut, 297
 dparam.lower_obj_cut_finite_trh, 298
 dparam.mio_disable_term_time, 298
 dparam.mio_max_time, 298
 dparam.mio_near_tol_abs_gap, 298
 dparam.mio_near_tol_rel_gap, 298
 dparam.mio_rel_gap_const, 299
 dparam.mio_tol_abs_gap, 299
 dparam.mio_tol_abs_relax_int, 299
 dparam.mio_tol_feas, 299
 dparam.mio_tol_rel_dual_bound_improvement, 299
 dparam.mio_tol_rel_gap, 299
 dparam.optimizer_max_time, 299
 dparam.presolve_tol_abs_lindep, 300
 dparam.presolve_tol_aij, 300
 dparam.presolve_tol_rel_lindep, 300
 dparam.presolve_tol_s, 300
 dparam.presolve_tol_x, 300
 dparam.qcqp_reformulate_rel_drop_tol, 300
 dparam.semidefinite_tol_approx, 300
 dparam.sim_lu_tol_rel_piv, 300
 dparam.simplex_abs_tol_piv, 301
 dparam.upper_obj_cut, 301
 dparam.upper_obj_cut_finite_trh, 301
 Integer parameters, 301
 iparam.ana_sol_basis, 301
 iparam.ana_sol_print_violated, 301
 iparam.auto_sort_a_before_opt, 301
 iparam.auto_update_sol_info, 302
 iparam.basis_solve_use_plus_one, 302
 iparam.bi_clean_optimizer, 302
 iparam.bi_ignore_max_iter, 302
 iparam.bi_ignore_num_error, 302
 iparam.bi_max_iterations, 302
 iparam.cache_license, 302
 iparam.check_convexity, 303
 iparam.compress_statfile, 303
 iparam.infeas_generic_names, 303
 iparam.infeas_prefer_primal, 303
 iparam.infeas_report_auto, 303
 iparam.infeas_report_level, 303
 iparam.intpnt_basis, 303
 iparam.intpnt_diff_step, 304
 iparam.intpnt_hotstart, 304
 iparam.intpnt_max_iterations, 304
 iparam.intpnt_max_num_cor, 304
 iparam.intpnt_max_num_refinement_steps, 304
 iparam.intpnt_multi_thread, 304
 iparam.intpnt_off_col_trh, 304
 iparam.intpnt_order_method, 305
 iparam.intpnt_regularization_use, 305
 iparam.intpnt_scaling, 305
 iparam.intpnt_solve_form, 305
 iparam.intpnt_starting_point, 305
 iparam.license_debug, 305
 iparam.license_pause_time, 305
 iparam.license_suppress_expire_wrns, 305
 iparam.license_trh_expiry_wrn, 306
 iparam.license_wait, 306
 iparam.log, 306
 iparam.log_ana_pro, 306
 iparam.log_bi, 306
 iparam.log_bi_freq, 306
 iparam.log_check_convexity, 306
 iparam.log_cut_second_opt, 307
 iparam.log_expand, 307
 iparam.log_feas_repair, 307
 iparam.log_file, 307
 iparam.log_infeas_ana, 307
 iparam.log_intpnt, 307
 iparam.log_mio, 308
 iparam.log_mio_freq, 308
 iparam.log_order, 308
 iparam.log_presolve, 308
 iparam.log_response, 308
 iparam.log_sensitivity, 308
 iparam.log_sensitivity_opt, 308
 iparam.log_sim, 309
 iparam.log_sim_freq, 309
 iparam.log_sim_minior, 309
 iparam.log_storage, 309
 iparam.max_num_warnings, 309
 iparam.mio_branch_dir, 309
 iparam.mio_construct_sol, 309
 iparam.mio_cut_clique, 309
 iparam.mio_cut_cmir, 310
 iparam.mio_cut_gmi, 310
 iparam.mio_cut_implied_bound, 310
 iparam.mio_cut_knapsack_cover, 310
 iparam.mio_cut_selection_level, 310
 iparam.mio_heuristic_level, 311
 iparam.mio_max_num_branches, 311
 iparam.mio_max_num_relaxs, 311
 iparam.mio_max_num_solutions, 311
 iparam.mio_mode, 311

iparam.mio_mt_user_cb, 311
iparam.mio_node_optimizer, 312
iparam.mio_node_selection, 312
iparam.mio_perspective_reformulate, 312
iparam.mio_probing_level, 312
iparam.mio_rins_max_nodes, 312
iparam.mio_root_optimizer, 312
iparam.mio_root_repeat_presolve_level, 312
iparam.mio_vb_detection_level, 313
iparam.mt_spincount, 313
iparam.num_threads, 313
iparam.opf_max_terms_per_line, 313
iparam.opf_write_header, 313
iparam.opf_write_hints, 313
iparam.opf_write_parameters, 313
iparam.opf_write_problem, 314
iparam.opf_write_sol_bas, 314
iparam.opf_write_sol_itg, 314
iparam.opf_write_sol_itr, 314
iparam.opf_write_solutions, 314
iparam.optimizer, 314
iparam.param_read_case_name, 314
iparam.param_read_ign_error, 315
iparam.presolve_eliminator_max_fill, 315
iparam.presolve_eliminator_max_num_tries, 315
iparam.presolve_level, 315
iparam.presolve_lindep_abs_work_trh, 315
iparam.presolve_lindep_rel_work_trh, 315
iparam.presolve_lindep_use, 315
iparam.presolve_max_num_reductions, 315
iparam.presolve_use, 316
iparam.primal_repair_optimizer, 316
iparam.read_data_compressed, 316
iparam.read_data_format, 316
iparam.read_debug, 316
iparam.read_keep_free_con, 316
iparam.read_lp_drop_new_vars_in_bou, 316
iparam.read_lp_quoted_names, 317
iparam.read_mps_format, 317
iparam.read_mps_width, 317
iparam.read_task_ignore_param, 317
iparam.remove_unused_solutions, 317
iparam.sensitivity_all, 317
iparam.sensitivity_optimizer, 317
iparam.sensitivity_type, 318
iparam.sim_basis_factor_use, 318
iparam.sim_degen, 318
iparam.sim_dual_crash, 318
iparam.sim_dual_phaseone_method, 318
iparam.sim_dual_restrict_selection, 318
iparam.sim_dual_selection, 318
iparam.sim_exploit_dupvec, 319
iparam.sim_hotstart, 319
iparam.sim_hotstart_lu, 319
iparam.sim_max_iterations, 319
iparam.sim_max_num_setbacks, 319
iparam.sim_non_singular, 319
iparam.sim_primal_crash, 319
iparam.sim_primal_phaseone_method, 320
iparam.sim_primal_restrict_selection, 320
iparam.sim_primal_selection, 320
iparam.sim_refactor_freq, 320
iparam.sim_reformulation, 320
iparam.sim_save_lu, 320
iparam.sim_scaling, 320
iparam.sim_scaling_method, 321
iparam.sim_solve_form, 321
iparam.sim_stability_priority, 321
iparam.sim_switch_optimizer, 321
iparam.sol_filter_keep_basic, 321
iparam.sol_filter_keep_ranged, 321
iparam.sol_read_name_width, 321
iparam.sol_read_width, 321
iparam.solution_callback, 322
iparam.timing_level, 322
iparam.write_bas_constraints, 322
iparam.write_bas_head, 322
iparam.write_bas_variables, 322
iparam.write_data_compressed, 322
iparam.write_data_format, 322
iparam.write_data_param, 322
iparam.write_free_con, 323
iparam.write_generic_names, 323
iparam.write_generic_names_io, 323
iparam.write_ignore_incompatible_items, 323
iparam.write_int_constraints, 323
iparam.write_int_head, 323
iparam.write_int_variables, 324
iparam.write_lp_full_obj, 324
iparam.write_lp_line_width, 324
iparam.write_lp_quoted_names, 324
iparam.write_lp_strict_format, 324
iparam.write_lp_terms_per_line, 324
iparam.write_mps_format, 324
iparam.write_mps_int, 324
iparam.write_precision, 325
iparam.write_sol_barvariables, 325
iparam.write_sol_constraints, 325
iparam.write_sol_head, 325
iparam.write_sol_ignore_invalid_names, 325
iparam.write_sol_variables, 325
iparam.write_task_inc_sol, 325
iparam.write_xml_mode, 325
String parameters, 326
sparam.bas_sol_file_name, 326
sparam.data_file_name, 326
sparam.debug_file_name, 326
sparam.int_sol_file_name, 326
sparam.itr_sol_file_name, 326
sparam.mio_debug_string, 326
sparam.param_comment_sign, 326
sparam.param_read_file_name, 326
sparam.param_write_file_name, 326
sparam.read_mps_bou_name, 327
sparam.read_mps_obj_name, 327

sparam.read_mps_ran_name, 327
 sparam.read_mps_rhs_name, 327
 sparam.remote_access_token, 327
 sparam.sensitivity_file_name, 327
 sparam.sensitivity_res_file_name, 327
 sparam.sol_filter_xc_low, 327
 sparam.sol_filter_xc_upr, 327
 sparam.sol_filter_xx_low, 328
 sparam.sol_filter_xx_upr, 328
 sparam.stat_file_name, 328
 sparam.stat_key, 328
 sparam.stat_name, 328
 sparam.write_lp_gen_var_name, 328

Response codes

Termination, 328

rescode.ok, 328

rescode.trm_internal, 329

rescode.trm_internal_stop, 329

rescode.trm_max_iterations, 328

rescode.trm_max_num_setbacks, 329

rescode.trm_max_time, 329

rescode.trm_mio_near_abs_gap, 329

rescode.trm_mio_near_rel_gap, 329

rescode.trm_mio_num_branches, 329

rescode.trm_mio_num_relaxes, 329

rescode.trm_num_max_num_int_solutions, 329

rescode.trm_numerical_problem, 329

rescode.trm_objective_range, 329

rescode.trm_stall, 329

rescode.trm_user_callback, 329

Warnings, 330

rescode.wrn_ana_almost_int_bounds, 332

rescode.wrn_ana_c_zero, 332

rescode.wrn_ana_close_bounds, 332

rescode.wrn_ana_empty_cols, 332

rescode.wrn_ana_large_bounds, 332

rescode.wrn_construct_invalid_sol_itg, 332

rescode.wrn_construct_no_sol_itg, 332

rescode.wrn_construct_solution_infeas, 332

rescode.wrn_dropped_nz_qobj, 330

rescode.wrn_duplicate_barvariable_names,
332

rescode.wrn_duplicate_cone_names, 332

rescode.wrn_duplicate_constraint_names, 332

rescode.wrn_duplicate_variable_names, 332

rescode.wrn_eliminator_space, 331

rescode.wrn_empty_name, 331

rescode.wrn_ignore_integer, 330

rescode.wrn_incomplete_linear_dependency_check,
331

rescode.wrn_large_aij, 330

rescode.wrn_large_bound, 330

rescode.wrn_large_cj, 330

rescode.wrn_large_con_fx, 330

rescode.wrn_large_lo_bound, 330

rescode.wrn_large_up_bound, 330

rescode.wrn_license_expire, 331

rescode.wrn_license_feature_expire, 331

rescode.wrn_license_server, 331

rescode.wrn_lp_drop_variable, 330

rescode.wrn_lp_old_quad_format, 330

rescode.wrn_mio_infeasible_final, 330

rescode.wrn_mps_split_bou_vector, 330

rescode.wrn_mps_split_ran_vector, 330

rescode.wrn_mps_split_rhs_vector, 330

rescode.wrn_name_max_len, 330

rescode.wrn_no_dualizer, 332

rescode.wrn_no_global_optimizer, 330

rescode.wrn_no_nonlinear_function_write,
331

rescode.wrn_nz_in_upr_tri, 330

rescode.wrn_open_param_file, 330

rescode.wrn_param_ignored_cmio, 331

rescode.wrn_param_name_dou, 331

rescode.wrn_param_name_int, 331

rescode.wrn_param_name_str, 331

rescode.wrn_param_str_value, 331

rescode.wrn_presolve_outofspace, 331

rescode.wrn_quad_cones_with_root_fixed_at_zero,
332

rescode.wrn_rquad_cones_with_root_fixed_at_zero,
332

rescode.wrn_sol_file_ignored_con, 331

rescode.wrn_sol_file_ignored_var, 331

rescode.wrn_sol_filter, 330

rescode.wrn_spar_max_len, 330

rescode.wrn_sym_mat_large, 332

rescode.wrn_too_few_basis_vars, 331

rescode.wrn_too_many_basis_vars, 331

rescode.wrn_undef_sol_file_name, 330

rescode.wrn_using_generic_names, 331

rescode.wrn_write_changed_names, 332

rescode.wrn_write_discarded_cfix, 332

rescode.wrn_zero_aij, 330

rescode.wrn_zeros_in_sparse_col, 331

rescode.wrn_zeros_in_sparse_row, 331

Errors, 333

rescode.err_ad_invalid_codelist, 346

rescode.err_api_array_too_small, 346

rescode.err_api_cb_connect, 346

rescode.err_api_fatal_error, 346

rescode.err_api_internal, 346

rescode.err_arg_is_too_large, 339

rescode.err_arg_is_too_small, 339

rescode.err_argument_dimension, 338

rescode.err_argument_is_too_large, 347

rescode.err_argument_lenneq, 338

rescode.err_argument_perm_array, 341

rescode.err_argument_type, 338

rescode.err_bar_var_dim, 347

rescode.err_basis, 340

rescode.err_basis_factor, 344

rescode.err_basis_singular, 344

rescode.err_blank_name, 335

rescode.err_cannot_clone_nl, 345

rescode.err_cannot_handle_nl, 345
rescode.err_cbf_duplicate_acoord, 349
rescode.err_cbf_duplicate_bcoord, 349
rescode.err_cbf_duplicate_con, 348
rescode.err_cbf_duplicate_int, 349
rescode.err_cbf_duplicate_obj, 348
rescode.err_cbf_duplicate_objacoord, 349
rescode.err_cbf_duplicate_psdvar, 349
rescode.err_cbf_duplicate_var, 348
rescode.err_cbf_invalid_con_type, 349
rescode.err_cbf_invalid_domain_dimension,
349
rescode.err_cbf_invalid_int_index, 349
rescode.err_cbf_invalid_psdvar_dimension,
349
rescode.err_cbf_invalid_var_type, 349
rescode.err_cbf_no_variables, 348
rescode.err_cbf_no_version_specified, 348
rescode.err_cbf_obj_sense, 348
rescode.err_cbf_parse, 348
rescode.err_cbf_syntax, 348
rescode.err_cbf_too_few_constraints, 349
rescode.err_cbf_too_few_ints, 349
rescode.err_cbf_too_few_psdvar, 349
rescode.err_cbf_too_few_variables, 349
rescode.err_cbf_too_many_constraints, 348
rescode.err_cbf_too_many_ints, 349
rescode.err_cbf_too_many_variables, 348
rescode.err_cbf_unsupported, 349
rescode.err_con_q_not_nsd, 341
rescode.err_con_q_not_psd, 341
rescode.err_cone_index, 341
rescode.err_cone_overlap, 342
rescode.err_cone_overlap_append, 342
rescode.err_cone_rep_var, 342
rescode.err_cone_size, 342
rescode.err_cone_type, 342
rescode.err_cone_type_str, 342
rescode.err_data_file_ext, 334
rescode.err_dup_name, 335
rescode.err_duplicate_aij, 342
rescode.err_duplicate_barvariable_names,
347
rescode.err_duplicate_cone_names, 347
rescode.err_duplicate_constraint_names, 347
rescode.err_duplicate_variable_names, 347
rescode.err_end_of_file, 334
rescode.err_factor, 344
rescode.err_feasrepair_cannot_relax, 344
rescode.err_feasrepair_inconsistent_bound,
344
rescode.err_feasrepair_solving_relaxed, 344
rescode.err_file_license, 333
rescode.err_file_open, 334
rescode.err_file_read, 334
rescode.err_file_write, 334
rescode.err_final_solution, 344
rescode.err_first, 340
rescode.err_firsti, 341
rescode.err_firstj, 341
rescode.err_fixed_bound_values, 343
rescode.err_flexlm, 333
rescode.err_global_inv_conic_problem, 344
rescode.err_huge_aij, 342
rescode.err_huge_c, 342
rescode.err_identical_tasks, 346
rescode.err_in_argument, 338
rescode.err_index, 339
rescode.err_index_arr_is_too_large, 338
rescode.err_index_arr_is_too_small, 338
rescode.err_index_is_too_large, 338
rescode.err_index_is_too_small, 338
rescode.err_inf_dou_index, 338
rescode.err_inf_dou_name, 339
rescode.err_inf_int_index, 338
rescode.err_inf_int_name, 339
rescode.err_inf_lint_index, 338
rescode.err_inf_lint_name, 339
rescode.err_inf_type, 339
rescode.err_infeas_undefined, 347
rescode.err_infinite_bound, 342
rescode.err_int64_to_int32_cast, 346
rescode.err_internal, 345
rescode.err_internal_test_failed, 346
rescode.err_inv_aptre, 340
rescode.err_inv_bk, 340
rescode.err_inv_bkc, 340
rescode.err_inv_bkx, 340
rescode.err_inv_cone_type, 341
rescode.err_inv_cone_type_str, 341
rescode.err_inv_marki, 345
rescode.err_inv_markj, 345
rescode.err_inv_name_item, 341
rescode.err_inv_numi, 345
rescode.err_inv_numj, 345
rescode.err_inv_optimizer, 344
rescode.err_inv_problem, 344
rescode.err_inv_qcon_subi, 342
rescode.err_inv_qcon_subj, 342
rescode.err_inv_qcon_subk, 342
rescode.err_inv_qcon_val, 343
rescode.err_inv_qobj_subi, 342
rescode.err_inv_qobj_subj, 342
rescode.err_inv_qobj_val, 342
rescode.err_inv_sk, 340
rescode.err_inv_sk_str, 340
rescode.err_inv_skc, 340
rescode.err_inv_skn, 340
rescode.err_inv_skn, 340
rescode.err_inv_skn, 340
rescode.err_inv_var_type, 340
rescode.err_invalid_accmode, 345
rescode.err_invalid_aij, 344
rescode.err_invalid_ampl_stub, 346
rescode.err_invalid_barvar_name, 335
rescode.err_invalid_compression, 345
rescode.err_invalid_con_name, 335

```

rescode.err_invalid_cone_name, 335
rescode.err_invalid_file_format_for_cones,
    347
rescode.err_invalid_file_format_for_general_mps,
    347
rescode.err_invalid_file_format_for_sym_mat,
    347
rescode.err_invalid_file_name, 334
rescode.err_invalid_format_type, 341
rescode.err_invalid_idx, 339
rescode.err_invalid_iomode, 345
rescode.err_invalid_max_num, 339
rescode.err_invalid_name_in_sol_file, 337
rescode.err_invalid_obj_name, 335
rescode.err_invalid_objective_sense, 343
rescode.err_invalid_problem_type, 347
rescode.err_invalid_sol_file_name, 334
rescode.err_invalid_stream, 335
rescode.err_invalid_surplus, 341
rescode.err_invalid_sym_mat_dim, 347
rescode.err_invalid_task, 335
rescode.err_invalid_utf8, 345
rescode.err_invalid_var_name, 335
rescode.err_invalid_wchar, 345
rescode.err_invalid_whichsol, 339
rescode.err_json_data, 338
rescode.err_json_format, 337
rescode.err_json_missing_data, 338
rescode.err_json_number_overflow, 337
rescode.err_json_string, 337
rescode.err_json_syntax, 337
rescode.err_last, 340
rescode.err_lasti, 341
rescode.err_lastj, 341
rescode.err_lau_arg_k, 348
rescode.err_lau_arg_m, 348
rescode.err_lau_arg_n, 348
rescode.err_lau_arg_trans, 348
rescode.err_lau_arg_transa, 348
rescode.err_lau_arg_transb, 348
rescode.err_lau_arg_uplo, 348
rescode.err_lau_invalid_lower_triangular_matrix,
    348
rescode.err_lau_invalid_sparse_symmetric_matrix,
    348
rescode.err_lau_not_positive_definite, 348
rescode.err_lau_singular_matrix, 348
rescode.err_lau_unknown, 348
rescode.err_license, 333
rescode.err_license_cannot_allocate, 333
rescode.err_license_cannot_connect, 333
rescode.err_license_expired, 333
rescode.err_license_feature, 333
rescode.err_license_invalid_hostid, 333
rescode.err_license_max, 333
rescode.err_license_moseklm_daemon, 333
rescode.err_license_no_server_line, 334
rescode.err_license_no_server_support, 333
rescode.err_license_server, 333
rescode.err_license_server_version, 333
rescode.err_license_version, 333
rescode.err_link_file_dll, 334
rescode.err_living_tasks, 335
rescode.err_lower_bound_is_a_nan, 342
rescode.err_lp_dup_slack_name, 336
rescode.err_lp_empty, 336
rescode.err_lp_file_format, 337
rescode.err_lp_format, 337
rescode.err_lp_free_constraint, 337
rescode.err_lp_incompatible, 336
rescode.err_lp_invalid_con_name, 337
rescode.err_lp_invalid_var_name, 337
rescode.err_lp_write_conic_problem, 337
rescode.err_lp_write_geco_problem, 337
rescode.err_lu_max_num_tries, 345
rescode.err_max_len_is_too_small, 341
rescode.err_maxnumbarvar, 339
rescode.err_maxnumcon, 339
rescode.err_maxnumcone, 342
rescode.err_maxnumqnz, 339
rescode.err_maxnumvar, 339
rescode.err_mio_internal, 347
rescode.err_mio_invalid_node_optimizer, 349
rescode.err_mio_invalid_root_optimizer, 349
rescode.err_mio_no_optimizer, 344
rescode.err_missing_license_file, 333
rescode.err_mixed_conic_and_nl, 344
rescode.err_mps_cone_overlap, 336
rescode.err_mps_cone_repeat, 336
rescode.err_mps_cone_type, 336
rescode.err_mps_duplicate_q_element, 336
rescode.err_mps_file, 335
rescode.err_mps_inv_bound_key, 336
rescode.err_mps_inv_con_key, 336
rescode.err_mps_inv_field, 335
rescode.err_mps_inv_marker, 335
rescode.err_mps_inv_sec_name, 336
rescode.err_mps_inv_sec_order, 336
rescode.err_mps_invalid_obj_name, 336
rescode.err_mps_invalid_objsense, 336
rescode.err_mps_mul_con_name, 336
rescode.err_mps_mul_csec, 336
rescode.err_mps_mul_qobj, 336
rescode.err_mps_mul_qsec, 336
rescode.err_mps_no_objective, 336
rescode.err_mps_non_symmetric_q, 336
rescode.err_mps_null_con_name, 335
rescode.err_mps_null_var_name, 335
rescode.err_mps_splitting_var, 336
rescode.err_mps_tab_in_field2, 336
rescode.err_mps_tab_in_field3, 336
rescode.err_mps_tab_in_field5, 336
rescode.err_mps_undef_con_name, 335
rescode.err_mps_undef_var_name, 335
rescode.err_mul_a_element, 340
rescode.err_name_is_null, 345

```

rescode.err_name_max_len, 345
rescode.err_nan_in_blc, 343
rescode.err_nan_in_blx, 343
rescode.err_nan_in_buc, 343
rescode.err_nan_in_bux, 343
rescode.err_nan_in_c, 343
rescode.err_nan_in_double_data, 343
rescode.err_negative_append, 340
rescode.err_negative_surplus, 340
rescode.err_newer_dll, 334
rescode.err_noBars_for_solution, 347
rescode.err_no_barx_for_solution, 347
rescode.err_no_basis_sol, 344
rescode.err_no_dual_for_itg_sol, 345
rescode.err_no_dual_infeas_cer, 345
rescode.err_no_init_env, 335
rescode.err_no_optimizer_var_type, 344
rescode.err_no_primal_infeas_cer, 345
rescode.err_no_snx_for_bas_sol, 345
rescode.err_no_solution_in_callback, 345
rescode.err_non_unique_array, 347
rescode.err_nonconvex, 341
rescode.err_nonlinear_equality, 341
rescode.err_nonlinear_functions_not_allowed, 343
rescode.err_nonlinear_ranged, 341
rescode.err_nr_arguments, 338
rescode.err_null_env, 334
rescode.err_null_pointer, 335
rescode.err_null_task, 335
rescode.err_numconlim, 339
rescode.err_numvarlim, 339
rescode.err_obj_q_not_nsd, 341
rescode.err_obj_q_not_psd, 341
rescode.err_objective_range, 340
rescode.err_older_dll, 334
rescode.err_open_dl, 334
rescode.err_opf_format, 337
rescode.err_opf_new_variable, 337
rescode.err_opf_premature_eof, 337
rescode.err_optimizer_license, 333
rescode.err_overflow, 344
rescode.err_param_index, 338
rescode.err_param_is_too_large, 338
rescode.err_param_is_too_small, 338
rescode.err_param_name, 338
rescode.err_param_name_dou, 338
rescode.err_param_name_int, 338
rescode.err_param_name_str, 338
rescode.err_param_type, 338
rescode.err_param_value_str, 338
rescode.err_platform_not_licensed, 333
rescode.err_postsolve, 344
rescode.err_pro_item, 341
rescode.err_prob_license, 333
rescode.err_qcon_subi_too_large, 343
rescode.err_qcon_subi_too_small, 343
rescode.err_qcon_upper_triangle, 343
rescode.err_qobj_upper_triangle, 343
rescode.err_read_format, 335
rescode.err_read_lp_missing_end_tag, 337
rescode.err_read_lp_nonexisting_name, 337
rescode.err_remove_cone_variable, 342
rescode.err_repair_invalid_problem, 344
rescode.err_repair_optimization_failed, 344
rescode.err_sen_bound_invalid_lo, 346
rescode.err_sen_bound_invalid_up, 346
rescode.err_sen_format, 346
rescode.err_sen_index_invalid, 346
rescode.err_sen_index_range, 346
rescode.err_sen_invalid_regexp, 346
rescode.err_sen_numerical, 346
rescode.err_sen_solution_status, 346
rescode.err_sen_undef_name, 346
rescode.err_sen_unhandled_problem_type, 346
rescode.err_server_connect, 350
rescode.err_server_protocol, 350
rescode.err_server_status, 350
rescode.err_server_token, 350
rescode.err_size_license, 333
rescode.err_size_license_con, 333
rescode.err_size_license_intvar, 333
rescode.err_size_license_numcores, 346
rescode.err_size_license_var, 333
rescode.err_sol_file_invalid_number, 342
rescode.err_solitem, 339
rescode.err_solver_probtype, 340
rescode.err_space, 334
rescode.err_space_leaking, 335
rescode.err_space_no_info, 335
rescode.err_sym_mat_duplicate, 347
rescode.err_sym_mat_huge, 344
rescode.err_sym_mat_invalid, 344
rescode.err_sym_mat_invalid_col_index, 347
rescode.err_sym_mat_invalid_row_index, 347
rescode.err_sym_mat_invalid_value, 347
rescode.err_sym_mat_not_lower_tringular, 347
rescode.err_task_incompatible, 345
rescode.err_task_invalid, 345
rescode.err_task_write, 345
rescode.err_thread_cond_init, 334
rescode.err_thread_create, 334
rescode.err_thread_mutex_init, 334
rescode.err_thread_mutex_lock, 334
rescode.err_thread_mutex_unlock, 334
rescode.err_toconic_constr_not_conic, 349
rescode.err_toconic_constr_q_not_psd, 349
rescode.err_toconic_constraint_fx, 349
rescode.err_toconic_constraint_ra, 349
rescode.err_toconic_objective_not_psd, 349
rescode.err_too_small_max_num_nz, 339
rescode.err_too_small_maxnumanz, 339
rescode.err_unb_step_size, 346
rescode.err_undef_solution, 340
rescode.err_undefined_objective_sense, 343

rescode.err_unhandled_solution_status, 347
rescode.err_unknown, 334
rescode.err_upper_bound_is_a_nan, 342
rescode.err_upper_triangle, 348
rescode.err_user_func_ret, 343
rescode.err_user_func_ret_data, 343
rescode.err_user_nlo_eval, 343
rescode.err_user_nlo_eval_hessubi, 343
rescode.err_user_nlo_eval_hessubj, 343
rescode.err_user_nlo_func, 343
rescode.err_whichitem_not_allowed, 339
rescode.err_whichsol, 339
rescode.err_write_lp_format, 337
rescode.err_write_lp_non_unique_name, 337
rescode.err_write_mps_invalid_name, 337
rescode.err_write_opf_invalid_var_name, 337
rescode.err_writing_file, 337
rescode.err_xml_invalid_problem_type, 346
rescode.err_y_is_undefined, 343

A

attaching
streams, 18

B

basis identification, 64, 129
basis type
sensitivity analysis, 155

BLAS, 71
DOT, 72

bound
constraint, 14, 113
linear optimization, 14
variable, 14, 113

C

callback, 61
CBF format, 406
certificate
dual, 115, 118, 119, 121
primal, 115, 117, 119
Cholesky factorization, 73
CLASSPATH, 7
column ordered
matrix format, 166
complementarity, 114
cone
dual, 117
quadratic, 22, 116
rotated quadratic, 22, 116
semidefinite, 26, 118
conic optimization, 22, 116
infeasibility, 117
interior-point, 133
termination criteria, 134
conic problem
example, 23
conic quadratic optimization, 22
Conic quadratic reformulation, 76
constraint
bound, 14, 113
linear optimization, 14
matrix, 14, 113, 121
quadratic, 120
constraints
lower limit, 121

upper limit, 121
convex interior-point
optimizers, 137

cqol
example, 23

creating
env, 12
task, 12

cut, 139

D

decision
variables, 121

defining
objective, 18

determinism, 126

DOT
BLAS, 72

dual
certificate, 115, 118, 119, 121
cone, 117
feasible, 114
infeasible, 114, 115, 118, 119, 121
problem, 113, 117, 118
variable, 114, 117

duality
conic, 117
gap, 114
linear, 113
semidefinite, 118

dualizer, 124

E

eliminator, 124

env
creating, 12
initializing, 12

environment variable
CLASSPATH, 7

example
conic problem, 23
cqol, 23
lo1, 18
lo2, 20
qol, 31, 32
quadratic constraints, 32
quadratic objective, 31

- simple, 12
- F
- factor model, 97
- fatal error, 108
- feasible
 - dual, 114
 - primal, 113, 127, 134
 - problem, 113
- format
 - CBF, 406
 - json, 421
 - LP, 380
 - MPS, 385
 - OPF, 396
 - OSiL, 420
 - sol, 429
 - task, 421
- full
 - vector format, 165
- G
- gap
 - duality, 114
- H
- hot-start, 131
- I
- infeasibility, 115, 117, 119
 - conic optimization, 117
 - linear optimization, 115
 - semidefinite, 119
- infeasible, 146
 - dual, 114, 115, 118, 119, 121
 - primal, 113, 115, 117, 119, 127, 134
 - problem, 113, 115, 117, 119
- infeasible problems, 146
- information item, 61
- initializing
 - env, 12
- installation, 6
 - nmake (command), 8, 9
 - requirements, 6
 - troubleshooting, 6
- integer
 - optimizer, 138
 - variable, 38
- integer feasible
 - solution, 140
- integer optimization, 38, 138
 - cut, 139
 - delayed termination criteria, 140
 - initial solution, 40
 - objective bound, 139
 - optimality gap, 141
 - relaxation, 139
 - termination criteria, 140
 - tolerance, 140
- integer optimizer
 - logging, 141
- interior-point
 - conic optimization, 133
 - linear optimization, 126
 - logging, 130, 136
 - optimizer, 126, 133
 - termination criteria, 128, 134
- interior-point optimizer, 137
- J
- json format, 421
- L
- LAPACK, 71
- leak, 108
- license, 85
- linear
 - objective, 18
- linear constraint matrix, 14
- linear dependency, 124
- linear optimization, 14, 113
 - bound, 14
 - constraint, 14
 - infeasibility, 115
 - interior-point, 126
 - objective, 14
 - simplex, 131
 - termination criteria, 128, 131
 - variable, 14
- linearity interval, 154
- lo1
 - example, 18
- lo2
 - example, 20
- logging
 - integer optimizer, 141
 - interior-point, 130, 136
 - optimizer, 130, 132, 136
 - simplex, 132
- lower limit
 - constraints, 121
 - variables, 121
- LP format, 380
- M
- Markowitz
 - model, 87
- Markowitz model
 - portfolio optimization, 87
- matrix
 - constraint, 14, 113, 121
 - semidefinite, 26
 - symmetric, 26
- matrix format
 - column ordered, 166
 - row ordered, 166

- triplets, 166
- memory leak, 108
- MIP, *see* integer optimization
- mixed-integer, *see* integer
- mixed-integer optimization, *see* integer optimization
- model
 - Markowitz, 87
 - portfolio optimization, 87
- MPS format, 385
 - free, 396
- N**
- near-optimal
 - solution, 129, 136, 140
- numerical issues
 - presolve, 124
 - scaling, 125
 - simplex, 132
- O**
- objective, 113
 - defining, 18
 - linear, 18
 - linear optimization, 14
- objective bound, 139
- objective vector, 121
- OPF
 - writing, 12
- OPF format, 396
- optimal
 - solution, 114
- optimality gap, 141
- optimization
 - conic quadratic, 116
 - linear, 14, 113
 - semidefinite, 118
- optimizer
 - determinism, 126
 - integer, 138
 - interior-point, 126, 133
 - interrupt, 61
 - logging, 130, 132, 136
 - parallelization, 125
 - selection, 124, 126
 - simplex, 131
- optimizers
 - convex interior-point, 137
- OSiL format, 420
- P**
- pair sensitivity analysis
 - optimal partition type, 155
- parallelization, 125
- parameter
 - simplex, 132
- portfolio optimization
 - model, 87
- positive semidefinite, 31
- presolve, 123
 - eliminator, 124
 - linear dependency check, 124
 - numerical issues, 124
- primal
 - certificate, 115, 117, 119
 - feasible, 113, 127, 134
 - infeasible, 113, 115, 117, 119, 127, 134
 - problem, 113, 117, 118
 - solution, 113
- primal-dual
 - problem, 126, 133
 - solution, 114
- problem
 - dual, 113, 117, 118
 - feasible, 113
 - infeasible, 113, 115, 117, 119
 - primal, 113, 117, 118
 - primal-dual, 126, 133
 - unbounded, 115
- Q**
- qo1
 - example, 31, 32
- quadratic
 - constraint, 120
- quadratic cone, 22, 116
- quadratic constraints
 - example, 32
- quadratic objective
 - example, 31
- quadratic optimization, 120
- quality
 - solution, 141
- R**
- relaxation, 139
- response handling, 42
- rotated quadratic cone, 22, 116
- row ordered
 - matrix format, 166
- S**
- scaling, 125
- scopt, 376
- semidefinite
 - cone, 26, 118
 - infeasibility, 119
 - matrix, 26
 - variable, 26, 118
- semidefinite optimization, 26, 118
- sensitivity analysis, 153
 - basis type, 155
- separable convex optimization, 57
- shadow price, 154
- simple
 - example, 12

- simplex
 - linear optimization, 131
 - logging, 132
 - numerical issues, 132
 - optimizer, 131
 - parameter, 132
 - termination criteria, 131
 - sol format, 429
 - solution
 - file format, 429
 - integer feasible, 140
 - near-optimal, 129, 136, 140
 - optimal, 114
 - primal, 113
 - primal-dual, 114
 - quality, 141
 - status, 17
 - solution summary, 50, 53
 - solving linear system, 68
 - sparse
 - vector format, 166
 - sparse vector, 166
 - status
 - solution, 17
 - Stream class, 109
 - streams
 - attaching, 18
 - symmetric
 - matrix, 26
- ## T
- task
 - creating, 12
 - task format, 421
 - termination criteria, 61
 - conic optimization, 134
 - delayed, 140
 - integer optimization, 140
 - interior-point, 128, 134
 - linear optimization, 128, 131
 - simplex, 131
 - tolerance, 129, 136, 140
 - thread, 125
 - time limit, 61
 - tolerance
 - integer optimization, 140
 - termination criteria, 129, 136, 140
 - triplets
 - matrix format, 166
 - troubleshooting
 - installation, 6
- ## U
- unbounded
 - problem, 115
 - upper limit
 - constraints, 121
 - variables, 122
 - user callback, *see* callback
- ## V
- variable, 113
 - bound, 14, 113
 - dual, 114, 117
 - integer, 38
 - linear optimization, 14
 - semidefinite, 26, 118
 - variables
 - decision, 121
 - lower limit, 121
 - upper limit, 122
 - vector format
 - full, 165
 - sparse, 166
- ## W
- writing
 - OPF, 12