MOSEK Optimization Toolbox for MATLAB

Release 9.0.81(BETA)

MOSEK ApS

15 March 2019

Contents

Chapter 1

Introduction

The **MOSEK** Optimization Suite 9.0.81(BETA) is a powerful software package capable of solving large-scale optimization problems of the following kind:

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

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

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

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

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

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

$$Ax - b \geq 0.$$

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

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

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

1.1 Why the Optimization Toolbox for MATLAB?

The Optimization Toolbox for MATLAB provides access to most of the functionality of **MOSEK** from a MATLAB environment. In addition the toolbox includes functions that replace functions from the MATLAB optimization toolbox available from MathWorks.

The Optimization Toolbox for MATLAB provides access to:

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

as well as to additional functions for:

- problem analysis,
- sensitivity analysis,
- infeasibility diagnostics.

Chapter 2

Contact Information

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

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

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

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

Chapter 3

License Agreement

Before using the **MOSEK** software, please read the license agreement available in the distribution at `<MSKHOME>/mosek/9.0/mosek-eula.pdf` or on the **MOSEK** website https://mosek.com/products/license-agreement.

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

zlib

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

Listing 3.1: *zlib* license.

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

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

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

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

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

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

fplib

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

Listing 3.2: *fplib* license.

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

(continues on next page)

(continued from previous page)

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

Zstandard

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

Listing 3.3: *Zstandard* license.

```
BSD License

For Zstandard software

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

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

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

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

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

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

Chapter 4

Installation

In this section we discuss how to install and setup the **MOSEK** Optimization Toolbox for MATLAB.

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 Optimization Toolbox for MATLAB can be used with MATLAB version R2015a or newer.

Locating files in the MOSEK Optimization Suite

The relevant files of the Optimization Toolbox for MATLAB are organized as reported in Table 4.1.

Table 4.1: Relevant files for the Optimization Toolbox for MATLAB.

Relative Path	Description	Label
`<MSKHOME>/mosek/9.0/toolbox/r2015a`	Toolbox	`<TOOLBOXDIR>`
`<MSKHOME>/mosek/9.0/toolbox/r2015aom`	Toolbox (without overloading)	`<TOOLBOXOMDIR>`
`<MSKHOME>/mosek/9.0/toolbox/examples`	Examples	`<EXDIR>`
`<MSKHOME>/mosek/9.0/toolbox/data`	Additional data	`<MISCDIR>`

where `<MSKHOME>` is the folder in which the **MOSEK** Optimization Suite has been installed.

Setting up the paths

To use Optimization Toolbox for MATLAB the path to the toolbox directory must be added via the `addpath` command in MATLAB. Use the command

```
addpath <MSKHOME>/mosek/9.0/toolbox/r2015a
```

or, if you do not want to overload functions such as *linprog* and *quadprog* from the MATLAB Optimization Toolbox with their **MOSEK** versions, then write

```
addpath <MSKHOME>/mosek/9.0/toolbox/r2015aom
```

On the Windows platform the relevant paths are

```
addpath <MSKHOME>\mosek\9.0\toolbox\r2015a
addpath <MSKHOME>\mosek\9.0\toolbox\r2015aom
```

Alternatively, the path to Optimization Toolbox for MATLAB may be set from the command line or it can be added to MATLAB permanently using the configuration file `startup.m` or from the FileSet Path menu item. We refer to MATLAB documentation for details.

4.1 Testing the installation

You can verify that Optimization Toolbox for MATLAB works by executing

```
mosekdiag
```

in MATLAB. This should produce a message similar to this:

```
mosekopt: /home/username/mosek/current/toolbox/r2015a/mosekopt.mexa64
Found MOSEK version : major(9), minor(0), revision(37)
mosekopt is working correctly.
```

If you only want to use Optimization Toolbox for MATLAB then warnings about non-availability of the command-line interface can be ignored.

More advanced debug information can be obtained with:

```
mosekopt('debug(10)')
```

4.1.1 Troubleshooting

Missing library files such as libmosek64.9.0.dylib or similar

If you are using Mac OS and get an error such as

```
Library not loaded: libmosek64.9.0.dylib
Referenced from:
/Users/.../mosek/9.0/toolbox/r2015a/mosekopt.mexmaci64
Reason: image not found.

Error in callmosek>doCall (line 224)
[res,sol] = mosekopt('minimize info',prob,param);
```

then most likely you did not run the **MOSEK** installation script `install.py` found in the `bin` directory. See also the Installation guide for details.

Windows, invalid MEX-file, cannot find shared libraries

If you are using Windows and get an error such as

```
Invalid MEX-file <MSKHOME>\Mosek\9.0\toolbox\r2015a\mosekopt.mexw64: The specified module␣
↪could not be found.
```

then MATLAB cannot load the **MOSEK** shared libraries, because the folder containing them is not in the system search path for DLLs. This can happen if **MOSEK** was installed manually and not using the MSI installer. The solution is to add the path `<MSKHOME>\mosek\9.0\tools\platform\<PLATFORM>\bin` to the system environment variable `PATH`. This can also be done per MATLAB session by using the `setenv` command in MATLAB before using **MOSEK**, for example:

```
setenv('PATH', [getenv('PATH') ';C:\Users\username\mosek\9.0\tools\platform\win64x86\bin']);
```

See also the Installation guide for details.

MOSEK does not see new license file

If you updated your license file but **MOSEK** does not detect it then restart MATLAB. **MOSEK** is caching the license and it will not notice the change in the license file on disk.

Undefined Function or Variable *mosekopt*

If you get the MATLAB error message

```
Undefined function or variable 'mosekopt'
```

you have not added the path to the Optimization Toolbox for MATLAB correctly as described above.

Invalid MEX-file

For certain versions of Windows and MATLAB, the path to MEX files cannot contain spaces. Therefore, if you have installed **MOSEK** in `C:\Program Files\Mosek` and get a MATLAB error similar to:

```
Invalid MEX-file <MSKHOME>\Mosek\9.0\toolbox\r2015a\mosekopt.mexw64
```

try installing **MOSEK** in a different directory, for example `C:\Users\<someuser>\` .

Output Arguments not assigned

If you encounter an error like

```
Error in ==> mosekpt at 1
function [r,res] = mosekopt(cmd,prob,param,callback)

Output argument "r" (and maybe others) not assigned during call to
"C:\Users\username\mosek\9.0\toolbox\r2015a\mosekopt.m>mosekopt".
```

then a mismatch between 32 and 64 bit versions of **MOSEK** and MATLAB is likely. From MATLAB type

```
which mosekopt
```

which (for a successful installation) should point to a MEX file,

```
<MSKHOME>\mosek\9.0\toolbox\r2015a\mosekopt.mexw64
```

and not to a MATLAB `.m` file,

```
<MSKHOME>\mosek\9.0\toolbox\r2015a\mosekopt.m
```

Chapter 5

Design Overview

5.1 Modeling

Optimization Toolbox for MATLAB is an interface for specifying optimization problems directly in matrix form. It means that an optimization problem such as:

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

or

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

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

The main characteristics of this interface are:

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

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

5.2 "Hello World!" in MOSEK

Here we present the most basic workflow pattern when using Optimization Toolbox for MATLAB.

Create a `prob` structure

Optimization problems using Optimization Toolbox for MATLAB are specified using a *`prob`* structure that describes the numerical data of the problem. In most cases it consists of matrices of floating-point numbers.

Retrieving the solutions

When the problem is set up, the optimizer is invoked with the call to *`mosekopt`*. The call will return a response and a structure containing the solution to all variables. See further details in Sec. 7.

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

Source code example

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

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

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

Listing 5.1: “Hello World!” in MOSEK

```
prob.a  =  sparse(0,1)  % 0 linear constraints, 1 variable
prob.c  =  [1.0]'       % Only objective coefficient
prob.blx=  [2.0]'       % Lower bound(s) on variable(s)
prob.bux=  [3.0]'       % Upper bound(s) on variable(s)

% Optimize
[r, res] = mosekopt('minimize', prob);

% Print answer
res.sol.itr.xx
```

Chapter 6

Optimization Tutorials

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

6.1 Linear Optimization

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

Minimize or maximize the objective function

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

subject to the linear constraints

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

and the bounds

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

The problem description consists of the following elements:

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

$$c = \left[\begin{array}{c} c_0 \\ \vdots \\ c_{n-1} \end{array} \right],$$

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

$$A = \left[\begin{array}{ccc} a_{0,0} & \cdots & a_{0,(n-1)} \\ \vdots & \cdots & \vdots \\ a_{(m-1),0} & \cdots & a_{(m-1),(n-1)} \end{array} \right],$$

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

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

6.1.1 Example LO1

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

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

under the bounds

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

Example: Linear optimization using `msklpopt`

A linear optimization problem such as (6.1) can be solved using the *msklpopt* function. The first step in solving the example is to setup the data for problem (6.1) i.e. the c, A, etc. Afterwards the problem is solved using an appropriate call to *msklpopt*.

Listing 6.1: Script implementing problem (6.1) using *msklpopt*.

```
function lo1()

c     = [3 1 5 1]';
a     = [[3 1 2 0];[2 1 3 1];[0 2 0 3]];
blc   = [30 15  -inf]';
buc   = [30 inf 25 ]';
blx   = zeros(4,1);
bux   = [inf 10 inf inf]';

[res] = msklpopt(c,a,blc,buc,blx,bux,[],'maximize');
sol   = res.sol;

% Interior-point solution.

sol.itr.xx'      % x solution.
sol.itr.sux'     % Dual variables corresponding to buc.
sol.itr.slx'     % Dual variables corresponding to blx.

% Basic solution.

sol.bas.xx'      % x solution in basic solution.
```

Please note that:

- Infinite bounds are specified using `-inf` and `inf`. Moreover, using `[]` for `bux`, `buc`, `blx` or `blc` means there are no bounds of the corresponding type.
- Retrieving different solution types is discussed in Sec. 7.1.

Example: Linear optimization using `mosekopt`

The function *msklpopt* is just a wrapper around the *mosekopt*, which is the main interface to **MOSEK** and is the only choice for more complicated problems, for instance with conic constraints. We demonstrate how to solve (6.1) directly with *mosekopt*. The following MATLAB code demonstrate how to set up the *prob* structure for the example (6.1) and solve the problem using *mosekopt*.

Listing 6.2: Script implementing problem (6.1) using *mosekopt*.

```
function lo2()
clear prob;

% Specify the c vector.
prob.c  = [3 1 5 1]';

% Specify a in sparse format.
subi   = [1 1 1 2 2 2 2 3 3];
subj   = [1 2 3 1 2 3 4 2 4];
valij  = [3 1 2 2 1 3 1 2 3];

prob.a = sparse(subi,subj,valij);

% Specify lower bounds of the constraints.
prob.blc = [30 15  -inf]';

% Specify  upper bounds of the constraints.
prob.buc = [30 inf 25 ]';

% Specify lower bounds of the variables.
prob.blx = zeros(4,1);

% Specify upper bounds of the variables.
prob.bux = [inf 10 inf inf]';

% Perform the optimization.
[r,res] = mosekopt('maximize',prob);

% Show the optimal x solution.
res.sol.bas.xx
```

Please note that

- A MATLAB structure named *prob* containing all the relevant problem data is defined.
- All fields of this structure are optional except `prob.a` which is required to be a **sparse** matrix. The dimension of this matrix determine the number of constraints and variables in the problem.
- Different parts of the solution can be accessed as described in Sec. 7.1.

Example: Linear optimization using *linprog*

MOSEK also provides a function *linprog* with a function of the same name from the MATLAB Optimization Toolbox. Consult Sec. 10.1 for details.

Listing 6.3: Script implementing problem (6.1) using *linprog*.

```
f     = - [3 1 5 1]';                         % minus because we maximize
A     = [[-2 -1 -3 -1]; [0 2 0 3]];
b     = [-15  25]';
Aeq   = [3 1 2 0];
beq   = 30;
l     = zeros(4,1);
u     = [inf 10 inf inf]';

% Example of setting options for linprog
% Get default options
opt = mskoptimset('');
% Turn on diagnostic output
opt = mskoptimset(opt,'Diagnostics','on');
```

(continues on next page)

(continued from previous page)

```
% Set a MOSEK option, in this case turn basic identification off.
opt = mskoptimset(opt,'MSK_IPAR_INTPNT_BASIS','MSK_OFF');
% Modefy a MOSEK parameter with double value
opt = mskoptimset(opt,'MSK_DPAR_INTPNT_TOL_INFEAS',1e-12);

[x,fval,exitflag,output,lambda] = linprog(f,A,b,Aeq,beq,l,u,opt);

x
fval
exitflag
output
lambda
```

6.2 Quadratic Optimization

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

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

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

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

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

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

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

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

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

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

6.2.1 Example: Quadratic Objective

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

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

The matrix formulation (6.4) has:

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

with the bounds:

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

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

Using `mosekopt`

In Listing 6.4 we show how to use *mosekopt* to solve problem (6.4). This is the preferred way.

Listing 6.4: How to solve problem (6.4) using *mosekopt*.

```
function qo2()

clear prob;

% c vector.
prob.c = [0 -1 0]';

% Define the data.

% First the lower triangular part of q in the objective
% is specified in a sparse format. The format is:
%
%   Q(prob.qosubi(t),prob.qosubj(t)) = prob.qoval(t), t=1,...,4

prob.qosubi = [ 1   3 2   3]';
prob.qosubj = [ 1   1 2   3]';
prob.qoval  = [ 2 -1 0.2 2]';

% a, the constraint matrix
subi  = ones(3,1);
subj  = 1:3;
valij = ones(3,1);

prob.a = sparse(subi,subj,valij);

% Lower bounds of constraints.
prob.blc  = [1.0]';

% Upper bounds of constraints.
prob.buc  = [inf]';

% Lower bounds of variables.
prob.blx  = sparse(3,1);

% Upper bounds of variables.
prob.bux = [];   % There are no bounds.

[r,res] = mosekopt('minimize',prob);

% Display return code.
fprintf('Return code: %d\n',r);
```

(continues on next page)

(continued from previous page)

```
% Display primal solution for the constraints.
res.sol.itr.xc'

% Display primal solution for the variables.
res.sol.itr.xx'
```

This sequence of commands looks much like the one that was used to solve the linear optimization example using *mosekopt* except that the definition of the Q matrix in `prob`. *mosekopt* requires that Q is specified in a sparse format. Indeed the vectors `qosubi`, `qosubj`, and `qoval` are used to specify the coefficients of Q in the objective using the principle

$$Q_{\texttt{qosubi(t)},\texttt{qosubj(t)}} = \texttt{qoval(t)}, \text{for} \quad t = 1, \ldots, \text{length}(\texttt{qosubi}).$$

An important observation is that due to Q being symmetric, only the lower triangular part of Q should be specified.

Using `mskqpopt`

In Listing 6.5 we show how to use *mskqpopt* to solve problem (6.4).

Listing 6.5: Function solving problem (6.4) using *mskqpopt*.

```
function qo1()

% Set up Q.
q     = [[2 0 -1];[0 0.2 0];[-1 0 2]];

% Set up the linear part of the problem.
c     = [0 -1 0]';
a     = ones(1,3);
blc   = [1.0];
buc   = [inf];
blx   = sparse(3,1);
bux   = [];

% Optimize the problem.
[res] = mskqpopt(q,c,a,blc,buc,blx,bux);

% Show the primal solution.
res.sol.itr.xx
```

It should be clear that the format for calling *mskqpopt* is very similar to calling *msklpopt* except that the Q matrix is included as the first argument of the call. Similarly, the solution can be inspected by viewing the `res.sol` field.

6.2.2 Example: Quadratic constraints

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

Consider the problem:

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

This is equivalent to

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

where

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

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

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

Setting up quadratic constraints

Please note that there are quadratic terms in both constraints. This problem can be solved using *mosekopt* as the following

Listing 6.6: Script implementing problem (6.5).

```
function qcqo1()
clear prob;

% Specify the linear objective terms.
prob.c      = [0, -1, 0];

% Specify the quadratic terms of the constraints.
prob.qcsubk = [1     1    1    1  ]';
prob.qcsubi = [1     2    3    3  ]';
prob.qcsubj = [1     2    3    1  ]';
prob.qcval  = [-2.0 -2.0 -0.2 0.2]';

% Specify the quadratic terms of the objective.
prob.qosubi = [1     2    3     3  ]';
prob.qosubj = [1     2    3     1  ]';
prob.qoval  = [2.0   0.2  2.0 -1.0]';

% Specify the linear constraint matrix
prob.a      = [1 1 1];

% Specify the lower bounds
prob.blc    = [1];
prob.blx    = zeros(3,1);

[r,res]     = mosekopt('minimize',prob);

% Display the solution.
fprintf('\nx:');
fprintf(' %-.4e',res.sol.itr.xx');
fprintf('\n||x||: %-.4e',norm(res.sol.itr.xx));
```

6.3 Conic Quadratic Optimization

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

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

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

MOSEK can solve conic quadratic optimization problems of the form

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

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

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

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

- Quadratic cone:

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

- Rotated quadratic cone:

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

For other types of cones supported by **MOSEK** see Sec. 6.4, Sec. 6.5, Sec. 6.6. Different cone types can appear together in one optimization problem.

For example, the following constraint:

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

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

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

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

6.3.1 Example CQO1

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

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

The linear constraints are specified as if the problem was a linear problem whereas the cones are specified using two index lists `cones.subptr` and `cones.sub` and list of cone-type identifiers `cones.type`. The elements of all the cones are listed in `cones.sub`, and `cones.subptr` specifies the index of the first element in `cones.sub` for each cone.

Listing 6.7 demonstrates how to solve the example (6.6) using **MOSEK**.

Listing 6.7: Script implementing problem (6.6).

```
function cqo1()

clear prob;

[r, res] = mosekopt('symbcon');
% Specify the non-conic part of the problem.

prob.c   = [0 0 0 1 1 1];
prob.a   = sparse([1 1 2 0 0 0]);
prob.blc = 1;
prob.buc = 1;
prob.blx = [0 0 0 -inf -inf -inf];
prob.bux = inf*ones(6,1);

% Specify the cones.

prob.cones.type   = [res.symbcon.MSK_CT_QUAD, res.symbcon.MSK_CT_RQUAD];
prob.cones.sub    = [4, 1, 2, 5, 6, 3];
prob.cones.subptr = [1, 4];
% The field 'type' specifies the cone types, i.e., quadratic cone
% or rotated quadratic cone. The keys for the two cone types are MSK_CT_QUAD
% and MSK_CT_RQUAD, respectively.
%
% The fields 'sub' and 'subptr' specify the members of the cones,
% i.e., the above definitions imply that
%   x(4) >= sqrt(x(1)^2+x(2)^2) and 2 * x(5) * x(6) >= x(3)^2.

% Optimize the problem.

[r,res]=mosekopt('minimize',prob);

% Display the primal solution.

res.sol.itr.xx'
```

6.4 Power Cone Optimization

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

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

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

MOSEK can solve conic optimization problems of the form

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

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

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

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

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

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

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

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

MOSEK also supports the dual power cone:

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

For other types of cones supported by **MOSEK** see Sec. 6.3, Sec. 6.5, Sec. 6.6. Different cone types can appear together in one optimization problem.

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

6.4.1 Example POW1

Consider the following optimization problem which involves powers of variables:

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

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

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

The linear constraints are specified as if the problem was a linear problem. The cone elements are specified using two index lists `cones.subptr` and `cones.sub`. Cone-type identifiers appear in `cones.type`, and the cone parameters α are in `cones.conepar`. The elements of all the cones are listed in `cones.sub`, and `cones.subptr` specifies the index of the first element in `cones.sub` for each cone.

Listing 6.8 demonstrates how to solve the example (6.7) using **MOSEK**. The solution is

```
[ 0.06389298  0.78308564  2.30604283 ]
```

Listing 6.8: Script implementing problem (6.7).

```
function pow1()

clear prob;

[r, res] = mosekopt('symbcon');
% Specify the non-conic part of the problem.

prob.c   = [-1 0 0 1 1 0];
prob.a   = [1 1 0.5 0 0 0];
prob.blc = [2.0];
prob.buc = [2.0];
prob.blx = [-inf -inf -inf -inf -inf 1.0];
prob.bux = [ inf  inf  inf  inf  inf 1.0];

% Specify the cones.
prob.cones.type   = [res.symbcon.MSK_CT_PPOW res.symbcon.MSK_CT_PPOW];
```

(continues on next page)

(continued from previous page)

```
prob.cones.conepar= [0.2 0.4];
prob.cones.sub    = [1 2 4 3 6 5];
prob.cones.subptr = [1 4];
% The field 'type' specifies the cone types, in this case power cones.
%
% The fields 'sub' and 'subptr' specify the members of the cones,
% i.e., the above definitions imply that
%   (x(1), x(2), x(4))  and (x(3), x(6), x(5))
% are cones.
%
% The field 'conepar' specifies the alpha cone parameters (exponents)

% Optimize the problem.

[r,res]=mosekopt('maximize',prob);

% Display the primal solution.

res.sol.itr.xx'
```

6.5 Conic Exponential Optimization

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

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

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

MOSEK can solve conic optimization problems of the form

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

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

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

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

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

MOSEK also supports the dual exponential cone:

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

For other types of cones supported by **MOSEK** see Sec. 6.3, Sec. 6.4, Sec. 6.6. Different cone types can appear together in one optimization problem.

For example, the following constraint:

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

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

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

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

6.5.1 Example CEO1

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

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

The conic form of (6.9) is:

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

The linear constraints are specified as if the problem was a linear problem whereas the cones are specified using two index lists `cones.subptr` and `cones.sub` and list of cone-type identifiers `cones.type`. The elements of all the cones are listed in `cones.sub`, and `cones.subptr` specifies the index of the first element in `cones.sub` for each cone.

Listing 6.9 demonstrates how to solve the example (6.9) using **MOSEK**.

Listing 6.9: Script implementing problem (6.9).

```
function ceo1()

clear prob;

[r, res] = mosekopt('symbcon');
% Specify the non-conic part of the problem.

prob.c   = [1 1 0];
prob.a   = sparse([1 1 1]);
prob.blc = 1;
prob.buc = 1;
prob.blx = [-inf -inf -inf];
prob.bux = [ inf  inf  inf];

% Specify the cones.

prob.cones.type   = [res.symbcon.MSK_CT_PEXP];
prob.cones.sub    = [1, 2, 3];
prob.cones.subptr = [1];
% The field 'type' specifies the cone types, in this case an exponential
% cone with key MSK_CT_PEXP.
%
% The fields 'sub' and 'subptr' specify the members of the cones,
% i.e., the above definitions imply that
%   x(1) >= x(2)*exp(x(3)/x(2))

% Optimize the problem.

[r,res]=mosekopt('minimize',prob);

% Display the primal solution.

res.sol.itr.xx'
```

6.6 Semidefinite Optimization

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

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

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

MOSEK can solve semidefinite optimization problems of the form

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

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

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

6.6.1 Example SDO1

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

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

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

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

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

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

subject to the two linear constraints

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

Listing 6.10 demonstrates how to solve this problem using **MOSEK**.

Listing 6.10: Code implementing problem (6.11).

```
function sdo1()
[r, res] = mosekopt('symbcon');

prob.c         = [1, 0, 0];

prob.bardim    = [3];
prob.barc.subj = [1, 1, 1, 1, 1];
prob.barc.subk = [1, 2, 2, 3, 3];
prob.barc.subl = [1, 1, 2, 2, 3];
prob.barc.val  = [2.0, 1.0, 2.0, 1.0, 2.0];

prob.blc = [1, 0.5];
prob.buc = [1, 0.5];

% It is a good practice to provide the correct
% dimmension of A as the last two arguments
% because it facilitates better error checking.
prob.a         = sparse([1, 2, 2], [1, 2, 3], [1, 1, 1], 2, 3);
prob.bara.subi = [1, 1, 1, 2, 2, 2, 2, 2, 2];
prob.bara.subj = [1, 1, 1, 1, 1, 1, 1, 1, 1];
prob.bara.subk = [1, 2, 3, 1, 2, 3, 2, 3, 3];
prob.bara.subl = [1, 2, 3, 1, 1, 1, 2, 2, 3];
prob.bara.val  = [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0];

prob.cones.type   = [res.symbcon.MSK_CT_QUAD];
prob.cones.sub    = [1, 2, 3];
prob.cones.subptr = [1];

[r,res] = mosekopt('minimize info',prob);

X = zeros(3);
X([1,2,3,5,6,9]) = res.sol.itr.barx;
X = X + tril(X,-1)';

x = res.sol.itr.xx;
```

The solution x is returned in `res.sol.itr.xx` and the numerical values of $\overline{X}_j$ are returned in `res.sol.barx`; the lower triangular part of each $\overline{X}_j$ is stacked column-by-column into an array, and each array is then concatenated forming a single array `res.sol.itr.barx` representing $\overline{X}_1, \ldots, \overline{X}_p$. Similarly, the dual semidefinite variables $\overline{S}_j$ are recovered through `res.sol.itr.bars`.

6.7 Affine conic constraints (new)

Optimization Toolbox for MATLAB can solve conic optimization problems in another format:

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

where $F \in \mathbb{R}^{k \times n}$ and $g \in \mathbb{R}^k$ specify an *affine conic constraint* of length (dimension) k. Usually $\mathcal{K}$ will be a product of basic cones corresponding to individual constraints.

In this tutorial we demonstrate how to use the affine conic format. It supports all types of basic cones available in **MOSEK** and can be combined with semidefinite variables as in Sec. 6.6.

6.7.1 Example AFFCO1

Consider the following simple optimization problem:

$$
\begin{array}{lrcl}
\text{maximize} & x_1^{1/3} + (x_1 + x_2 + 0.1)^{1/4} & & \\
\text{subject to} & (x_1 - 0.5)^2 + (x_2 - 0.6)^2 & \leq & 1, \\
 & x_1 - x_2 & \leq & 1.
\end{array} \tag{6.12}
$$

Adding auxiliary variables we convert this problem into an equivalent conic form:

$$
\begin{array}{lrcl}
\text{maximize} & t_1 + t_2 & & \\
\text{subject to} & (1, x_1 - 0.5, x_2 - 0.6) & \in & \mathcal{Q}^3, \\
 & (x_1, 1, t_1) & \in & \mathcal{P}_3^{1/3,2/3}, \\
 & (x_1 + x_2 + 0.1, 1, t_2) & \in & \mathcal{P}_3^{1/4,3/4}, \\
 & x_1 - x_2 & \leq & 1.
\end{array} \tag{6.13}
$$

Note that each of the vectors constrained to a cone is in a natural way an affine combination of the problem variables.

We first set up the linear part of the problem, including the number of variables, objective and all bounds precisely as in Sec. 6.1. Cones will be defined using the *cones* structure. We construct the matrices F, g for each of the three cones. For example, the constraint $(1, x_1 - 0.5, x_2 - 0.6) \in \mathcal{Q}^3$ is written in matrix form as

$$
\begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ t_1 \\ t_2 \end{bmatrix} + \begin{bmatrix} 1 \\ -0.5 \\ -0.6 \end{bmatrix} \in \mathcal{Q}^3.
$$

Below we set up the matrices and define the cone type as a quadratic cone of length 3:

```
% The quadratic cone
FQ = sparse([zeros(1,4); speye(2) zeros(2,2)]);
gQ = [1 -0.5 -0.6]';
cQ = [res.symbcon.MSK_CT_QUAD 3];
```

Next we demonstrate how to do the same for the second of the power cone constraints. Its affine representation is:

$$
\begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ t_1 \\ t_2 \end{bmatrix} + \begin{bmatrix} 0.1 \\ 1 \\ 0 \end{bmatrix} \in \mathcal{P}_3^{1/4,3/4}.
$$

The power cone is defined by its type, length, number of additional parameters (always equal to 2) and the exponents $\alpha, 1-\alpha$ appearing in the cone definition. In fact any pair of positive real numbers *proportional* to $(\alpha, 1-\alpha)$ may be used. They will be normalized to add up to 1:

```
% The power cone for (x_1+x_2+0.1, 1, t_2) \in POW3^(1/4,3/4)
FP2 = sparse([1 1 zeros(1,2); zeros(1,4); zeros(1,2) 0 1]);
gP2 = [0.1 1 0]';
cP2 = [res.symbcon.MSK_CT_PPOW 3 2 1.0 3.0];
```

Once affine conic descriptions of all cones are ready it remains to stack them vertically into the matrix F and vector g and concatenate the cone descriptions in one list. Below is the full code for problem (6.13).

Listing 6.11: Script implementing conic version of problem (6.12).

```
function affco1()

[rcode, res] = mosekopt('symbcon echo(0)');
prob = [];
```

(continues on next page)

(continued from previous page)

```
% Variables [x1; x2; t1; t2]
prob.c = [0, 0, 1, 1];

% Linear inequality x_1 - x_2 <= 1
prob.a = sparse([1, -1, 0, 0]);
prob.buc = 1;
prob.blc = [];

% The quadratic cone
FQ = sparse([zeros(1,4); speye(2) zeros(2,2)]);
gQ = [1 -0.5 -0.6]';
cQ = [res.symbcon.MSK_CT_QUAD 3];

% The power cone for (x_1, 1, t_1) \in POW3^(1/3,2/3)
FP1 = sparse([1 0 zeros(1,2); zeros(1,4); zeros(1,2) 1 0]);
gP1 = [0 1 0]';
cP1 = [res.symbcon.MSK_CT_PPOW 3 2 1/3 2/3];

% The power cone for (x_1+x_2+0.1, 1, t_2) \in POW3^(1/4,3/4)
FP2 = sparse([1 1 zeros(1,2); zeros(1,4); zeros(1,2) 0 1]);
gP2 = [0.1 1 0]';
cP2 = [res.symbcon.MSK_CT_PPOW 3 2 1.0 3.0];

% All cones
prob.f = [FQ; FP1; FP2];
prob.g = [gQ; gP1; gP2];
prob.cones = [cQ cP1 cP2];

[r, res] = mosekopt('maximize', prob);

res.sol.itr.pobjval
res.sol.itr.xx(1:2)
```

6.7.2 Example AFFCO2

Consider the following simple linear dynamical system. A point in $\mathbb{R}^n$ moves along a trajectory given by $z(t) = z(0)\exp(At)$, where $z(0)$ is the starting position and $A = \mathbf{Diag}(a_1, \ldots, a_n)$ is a diagonal matrix with $a_i < 0$. Find the time after which $z(t)$ is within euclidean distance d from the origin. Denoting the coordinates of the starting point by $z(0) = (z_1, \ldots, z_n)$ we can write this as an optimization problem in one variable t:

$$\begin{array}{lrcl} \text{minimize} & & & t \\ \text{subject to} & \sqrt{\sum_i (z_i \exp(a_i t))^2} & \leq & d, \end{array}$$

which can be cast into conic form as:

$$\begin{array}{lrcl} \text{minimize} & & & t \\ \text{subject to} & (d, z_1 y_1, \ldots, z_n y_n) & \in & \mathcal{Q}^{n+1}, \\ & (y_i, 1, a_i t) & \in & K_{\exp}, \; i = 1, \ldots, n, \end{array} \tag{6.14}$$

with variable vector $x = [t, y_1, \ldots, y_n]^T$.

We assemble all conic constraints in the form

$$Fx + g \in \mathcal{Q}^{n+1} \times (K_{\exp})^n.$$

For the conic quadratic constraint this representation is

$$\left[\begin{array}{cc} 0 & 0_n^T \\ 0_n & \mathbf{Diag}(z_1, \ldots, z_n) \end{array}\right] \left[\begin{array}{c} t \\ y \end{array}\right] + \left[\begin{array}{c} d \\ 0_n \end{array}\right] \in \mathcal{Q}^{n+1}.$$

For the i-th exponential cone we have

$$\begin{bmatrix} 0 & e_i^T \\ 0 & 0_n \\ a_i & 0_n \end{bmatrix} \begin{bmatrix} t \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \in K_{\exp},$$

where e_i denotes a vector of length n with a single 1 in position i.

Listing 6.12: Script implementing problem (6.14).

```
function t = firstHittingTime(n, z, a, d)

[rcode, res] = mosekopt('symbcon echo(0)');
prob = [];

% Variables [t, y1, ..., yn]
prob.a = sparse(0, n+1);
prob.c = [1 zeros(1,n)];

% Quadratic cone
FQ = diag([0; z]);
gQ = [d; zeros(n,1)];

% All exponential cones
FE = sparse([1:3:3*n    3:3:3*n], ...
            [2:n+1      ones(1,n)], ...
            [ones(1,n)  a']);
gE = repmat([0; 1; 0], n, 1);

% Assemble input data
prob.f = [FQ; FE];
prob.g = [gQ; gE];
prob.cones = [res.symbcon.MSK_CT_QUAD n+1 repmat([res.symbcon.MSK_CT_PEXP 3], 1, n)];

% Solve
[r, res] = mosekopt('minimize', prob);
t = res.sol.itr.xx(1)
```

6.8 Geometric Programming

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

$$\begin{array}{lll} \text{minimize} & f_0(x) & \\ \text{subject to} & f_i(x) \leq 1, & i = 1, \ldots, m, \\ & x_j > 0, & j = 1, \ldots, n, \end{array} \tag{6.15}$$

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

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

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

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

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

$$\log(\sum_k \exp(a_k^T y + b_k)) \leq 0 \tag{6.16}$$

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

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

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

6.8.1 Example GP1

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

$$\begin{array}{lrl} \text{maximize} & hwd & \\ \text{subject to} & 2(hw+hd) & \leq A_{\text{wall}}, \\ & wd & \leq A_{\text{floor}}, \\ & \alpha & \leq h/w \leq \beta, \\ & \gamma & \leq d/w \leq \delta. \end{array} \tag{6.17}$$

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

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

after which (6.17) becomes

$$\begin{array}{ll} \text{maximize} & x+y+z \\ \text{subject to} & \log(\exp(x+y+\log(2/A_{\text{wall}})) + \exp(x+z+\log(2/A_{\text{wall}}))) \leq 0, \\ & y+z \leq \log(A_{\text{floor}}), \\ & \log(\alpha) \leq x-y \leq \log(\beta), \\ & \log(\gamma) \leq z-y \leq \log(\delta). \end{array} \tag{6.18}$$

Next, we demonstrate how to implement a log-sum-exp constraint (6.16). It can be written as:

$$\begin{array}{ll} u_k \geq \exp(a_k^T y + b_k), & (\text{equiv. } (u_k, 1, a_k^T y + b_k) \in K_{\exp}), \\ \sum_k u_k = 1. & \end{array} \tag{6.19}$$

This presentation requires one extra variable u_k for each monomial appearing in the original posynomial constraint. It is natural to express the cone membership using an affine conic constraint (see Sec. 6.7). In this case:

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ u_1 \\ u_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ \log(2/A_{\text{wall}}) \\ 0 \\ 1 \\ \log(2/A_{\text{wall}}) \end{bmatrix} \in K_{\exp} \times K_{\exp}.$$

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

Listing 6.13: Source code solving problem (6.18).

```
[r,res] = mosekopt('symbcon');

% Input data
Awall = 200;
Afloor = 50;
alpha = 2;
beta = 10;
gamma = 2;
```

(continues on next page)

```
delta = 10;

% Objective
prob = [];
prob.c = [1, 1, 1, 0, 0]';

% Linear constraints:
% [ 0  0  0  1  1 ]                    == 1
% [ 0  1  1  0  0 ]                    <= log(Afloor)
% [ 1 -1  0  0  0 ]                    in [log(alpha), log(beta)]
% [ 0 -1  1  0  0 ]                    in [log(gamma), log(delta)]
%
prob.a = [ 0  0  0  1  1;
           0  1  1  0  0;
           1 -1  0  0  0;
           0 -1  1  0  0 ];

prob.blc = [ 1; -inf;        log(alpha); log(gamma) ];
prob.buc = [ 1; log(Afloor); log(beta);  log(delta) ];

prob.blx = [ -inf; -inf; -inf; -inf; -inf];
prob.bux = [ inf; inf; inf; inf; inf];

% The conic part FX+g \in Kexp x Kexp
%   x  y  z  u  v
% [ 0  0  0  1  0 ]    0
% [ 0  0  0  0  0 ]    1               in Kexp
% [ 1  1  0  0  0 ]    log(2/Awall)
%
% [ 0  0  0  0  1 ]    0
% [ 0  0  0  0  0 ]    1               in Kexp
% [ 1  0  1  0  0 ] +  log(2/Awall)
%
%
prob.f = sparse([0 0 0 1 0;
                 0 0 0 0 0;
                 1 1 0 0 0;
                 0 0 0 0 1;
                 0 0 0 0 0;
                 1 0 1 0 0]);

prob.g = [ 0; 1; log(2/Awall); 0; 1; log(2/Awall)];

prob.cones = [ res.symbcon.MSK_CT_PEXP, 3, res.symbcon.MSK_CT_PEXP, 3 ];

% Optimize and print results
[r,res]=mosekopt('maximize',prob);
exp(res.sol.itr.xx(1:3))
```

6.9 Integer Optimization

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

6.9.1 Example MILO1

We use the example

$$
\begin{array}{lccl}
\text{maximize} & x_0 + 0.64x_1 & & \\
\text{subject to} & 50x_0 + 31x_1 & \leq & 250, \\
& 3x_0 - 2x_1 & \geq & -4, \\
& x_0, x_1 \geq 0 & & \text{and integer}
\end{array}
\tag{6.20}
$$

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

The complete source for the example is listed in Listing 6.14.

Listing 6.14: How to solve problem (6.20).

```
function milo1()
clear prob
prob.c        = [1 0.64];
prob.a        = [[50 31];[3 -2]];
prob.blc      = [-inf -4];
prob.buc      = [250 inf];
prob.blx      = [0 0];
prob.bux      = [inf inf];

% Specify indexes of variables that are integer
% constrained.

prob.ints.sub = [1 2];

% Optimize the problem.
[r,res] = mosekopt('maximize',prob);

try
  % Display the optimal solution.
  res.sol.int
  res.sol.int.xx'
catch
  fprintf('MSKERROR: Could not get solution')
end
```

Please note that compared to a linear optimization problem with no integer-constrained variables:

- The `prob.ints.sub` field is used to specify the indexes of the variables that are integer-constrained.
- The optimal integer solution is returned in the `res.sol.int` MATLAB structure.

MOSEK also provides a wrapper for the *intlinprog* function found in the MATLAB optimization toolbox. This function solves linear problems wth integer variables; see the reference section for details.

6.9.2 Specifying an initial solution

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

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

$$
\begin{array}{ll}
\text{maximize} & 7x_0 + 10x_1 + x_2 + 5x_3 \\
\text{subject to} & x_0 + x_1 + x_2 + x_3 \leq 2.5 \\
 & x_0, x_1, x_2 \in \mathbb{Z} \\
 & x_0, x_1, x_2, x_3 \geq 0
\end{array}
\tag{6.21}
$$

Solution values can be set using the appropriate fields in the problem structure.

Listing 6.15: Implementation of problem (6.21) specifying an initial solution.

```
% Specify start guess for the integer variables.
prob.sol.int.xx  = [1 1 0 nan]';
```

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

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

The same information can be obtained from the API:

Listing 6.16: Retrieving information about usage of initial solution

```
  res.info.MSK_IINF_MIO_CONSTRUCT_SOLUTION
  res.info.MSK_DINF_MIO_CONSTRUCT_SOLUTION_OBJ
```

6.9.3 Example MICO1

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

$$
\begin{array}{ll}
\text{minimize} & x^2 + y^2 \\
\text{subject to} & x \geq e^y + 3.8, \\
 & x, y \text{ integer}.
\end{array}
\tag{6.22}
$$

The canonical conic formulation of (6.22) suitable for Optimization Toolbox for MATLAB is

$$
\begin{array}{lll}
\text{minimize} & t & \\
\text{subject to} & (t, x, y) \in \mathcal{Q}^3 & (t \geq \sqrt{x^2 + y^2}) \\
 & (x - 3.8, 1, y) \in K_{\exp} & (x - 3.8 \geq e^y) \\
 & x, y \text{ integer}, & \\
 & t \in \mathbb{R}. &
\end{array}
\tag{6.23}
$$

Listing 6.17: Implementation of problem (6.23).

```
[rcode, res] = mosekopt('symbcon echo(0)');
symbcon = res.symbcon;
clear prob

% The full variable is [t; x; y]
prob.c         = [1 0 0];
prob.a         = sparse(0,3);    % No constraints

% Conic part of the problem
prob.f = sparse([ eye(3);
                  0 1 0;
                  0 0 0;
                  0 0 1 ]);
prob.g = [0 0 0 -3.8 1 0]';
```

(continues on next page)

(continued from previous page)

```
prob.cones = [symbcon.MSK_CT_QUAD 3 symbcon.MSK_CT_PEXP 3];

% Specify indexes of variables that are integers
prob.ints.sub = [2 3];

% Optimize the problem.
[r,res] = mosekopt('minimize',prob);

try
  res.sol.int.xx(2:3)
catch
  fprintf('MSKERROR: Could not get solution')
end
```

Note that the conic constraints are described using the format $Fx + g \in \mathcal{K}$, that is as *affine conic constraints*. See Sec. 6.7 for details.

Error and solution status handling were omitted for readability.

6.10 Problem Modification and Reoptimization

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

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

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

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

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

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

6.10.1 Example: Production Planning

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

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

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

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

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

and

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

Code in Listing 6.18 loads and solves this problem.

Listing 6.18: Setting up and solving problem (6.24)

```
% Specify the c vector.
prob.c  = [1.5 2.5 3.0]';

% Specify a in sparse format.
subi   = [1 1 1 2 2 2 3 3 3];
subj   = [1 2 3 1 2 3 1 2 3];
valij  = [2 4 3 3 2 3 2 3 2];

prob.a = sparse(subi,subj,valij);

% Specify lower bounds of the constraints.
prob.blc = [-inf -inf -inf]';

% Specify  upper bounds of the constraints.
prob.buc = [100000 50000 60000]';

% Specify lower bounds of the variables.
prob.blx = zeros(3,1);

% Specify upper bounds of the variables.
prob.bux = [inf inf inf]';

% Perform the optimization.
param.MSK_IPAR_OPTIMIZER = 'MSK_OPTIMIZER_FREE_SIMPLEX';
[r,res] = mosekopt('maximize',prob,param);

% Show the optimal x solution.
res.sol.bas.xx
```

6.10.2 Changing the Linear Constraint Matrix

Suppose we want to change the time required for assembly of product 0 to 3 minutes. This corresponds to setting $a_{0,0} = 3$, which is done by directly modifying the A matrix of the problem, as shown below.

```
prob.a(1,1) = 3.0;
```

The problem now has the form:

$$\begin{array}{lccccccc} \text{maximize} & 1.5x_0 & + & 2.5x_1 & + & 3.0x_2 & & \\ \text{subject to} & 3x_0 & + & 4x_1 & + & 3x_2 & \leq & 100000, \\ & 3x_0 & + & 2x_1 & + & 3x_2 & \leq & 50000, \\ & 2x_0 & + & 3x_1 & + & 2x_2 & \leq & 60000, \end{array} \tag{6.25}$$

and

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

After this operation we can reoptimize the problem.

6.10.3 Appending Variables

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

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

This corresponds to creating a new variable x_3, appending a new column to the A matrix and setting a new term in the objective. We do this in Listing 6.19

Listing 6.19: How to add a new variable (column)

```
prob.c      = [prob.c;1.0];
prob.a      = [prob.a,sparse([4.0 0.0 1.0]')];
prob.blx    = [prob.blx; 0.0];
prob.bux    = [prob.bux; inf];
```

After this operation the new problem is:

$$
\begin{array}{lccccccccc}
\text{maximize} & 1.5x_0 & + & 2.5x_1 & + & 3.0x_2 & + & 1.0x_3 & & \\
\text{subject to} & 3x_0 & + & 4x_1 & + & 3x_2 & + & 4x_3 & \leq & 100000, \\
 & 3x_0 & + & 2x_1 & + & 3x_2 & & & \leq & 50000, \\
 & 2x_0 & + & 3x_1 & + & 2x_2 & + & 1x_3 & \leq & 60000,
\end{array}
\tag{6.26}
$$

and

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

6.10.4 Appending Constraints

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

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

This corresponds to adding the constraint

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

to the problem. This is done as follows.

Listing 6.20: Adding a new constraint.

```
prob.a      = [prob.a;sparse([1.0 2.0 1.0 1.0])];
prob.blc    = [prob.blc; -inf];
prob.buc    = [prob.buc; 30000.0];
```

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

6.10.5 Changing bounds

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

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

That means we would like to solve the problem:

$$
\begin{array}{lccccccccl}
\text{maximize} & 1.5x_0 & + & 2.5x_1 & + & 3.0x_2 & + & 1.0x_3 & & \\
\text{subject to} & 3x_0 & + & 4x_1 & + & 3x_2 & + & 4x_3 & \leq & 80000, \\
& 3x_0 & + & 2x_1 & + & 3x_2 & & & \leq & 40000, \\
& 2x_0 & + & 3x_1 & + & 2x_2 & + & 1x_3 & \leq & 50000, \\
& x_0 & + & 2x_1 & + & x_2 & + & x_3 & \leq & 22000.
\end{array} \tag{6.27}
$$

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

Listing 6.21: Change constraint bounds.

```
prob.buc      = [80000 40000 50000 22000]';
prob.sol      = res.sol;
[r,res] = mosekopt('maximize',prob,param);
res.sol.bas.xx
```

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

6.10.6 Advanced hot-start

In order to exploit the possibility of hot-starting the simplex algorithms it is necessary to pass the old basic solution when the modified problem is re-optimized. Without this operation the optimizer will simply start from scratch. Any subset of the basic solution may be provided, but to achieve the best results all fields of `res.sol.bas` should be present, that is `xx,xc,y,slx,sux,slc,suc,skx,skc`.

Listing 6.22: Passing the full basic solution.

```
% Reoptimize with changed coefficient
% Use previous solution to perform very simple hot-start.
% This part can be skipped, but then the optimizer will start
% from scratch on the new problem, i.e. without any hot-start.
prob.sol = [];
prob.sol.bas = res.sol.bas;
[r,res] = mosekopt('maximize',prob,param);
res.sol.bas.xx
```

If the dimensions of the problem (number of variables, constraints) have changed, the lengths of all fields have to be adjusted to be compatible with the reformulated problem. For example, here is an adjustment when adding a new variable:

Listing 6.23: Adjusting lengths in the solution fields related to variables.

```
% Reoptimize with a new variable and hot-start
% All parts of the solution must be extended to the new dimensions.
prob.sol = [];
prob.sol.bas = res.sol.bas;
prob.sol.bas.xx  = [prob.sol.bas.xx; 0.0];
prob.sol.bas.slx = [prob.sol.bas.slx; 0.0];
prob.sol.bas.sux = [prob.sol.bas.sux; 0.0];
prob.sol.bas.skx = [prob.sol.bas.skx; 'UN'];
[r,res] = mosekopt('maximize',prob,param);
res.sol.bas.xx
```

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

```
Optimizer  - hotstart               : yes
```

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

A more advanced discussion of hot-start is presented in Sec. 9.2.
For a more in-depth treatment see the following sections:

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

Chapter 7

Solver Interaction Tutorials

In this section we cover the interaction with the solver.

7.1 Accessing the solution

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

7.1.1 Solver termination

The optimizer provides a **response code** of type *rescode*, relevant for error handling. It indicates if any errors occurred in any phase of optimization (including processing input data). It will also indicate system-related errors (such as an out of memory error, licensing error etc.). The expected value for a successful optimization is *"MSK_RES_OK"*.

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

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

7.1.2 Available solutions

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

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

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

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

	Simplex optimizer	Interior-point optimizer	Mixed-integer optimizer
Linear problem	`res.sol.bas`	`res.sol.itr`	
Nonlinear continuous problem		`res.sol.itr`	
Problem with integer variables			`res.sol.int`

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

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

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

7.1.3 Problem and solution status

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

Problem status

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

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

Solution status

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

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

Problem and solution status can be found in the fields `prosta` and `solsta` of a solution structure *solution*, for instance `res.sol.itr.prosta`, `res.sol.itr.solsta` for the interior-point solution.

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

Typical status reports

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

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

Outcome	Problem status	Solution status
Optimal	*"MSK_PRO_STA_PRIM_AND_DUAL_FEAS"*	*"MSK_SOL_STA_OPTIMAL"*
Primal infeasible	*"MSK_PRO_STA_PRIM_INFEAS"*	*"MSK_SOL_STA_PRIM_INFEAS_CER"*
Dual infeasible (unbounded)	*"MSK_PRO_STA_DUAL_INFEAS"*	*"MSK_SOL_STA_DUAL_INFEAS_CER"*
Uncertain (stall, numerical issues, etc.)	*"MSK_PRO_STA_UNKNOWN"*	*"MSK_SOL_STA_UNKNOWN"*

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

Outcome	Problem status	Solution status
Integer optimal	*"MSK_PRO_STA_PRIM_FEAS"*	*"MSK_SOL_STA_INTEGER_OPTIMAL"*
Infeasible	*"MSK_PRO_STA_PRIM_INFEAS"*	*"MSK_SOL_STA_UNKNOWN"*
Integer feasible point	*"MSK_PRO_STA_PRIM_FEAS"*	*"MSK_SOL_STA_PRIM_FEAS"*
No conclusion	*"MSK_PRO_STA_UNKNOWN"*	*"MSK_SOL_STA_UNKNOWN"*

7.1.4 Retrieving solution values

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

- `res.sol.itr.pobjval`, `res.sol.itr.dobjval` — the primal and dual objective value.
- `res.sol.itr.xx` — solution values for the variables.
- `res.sol.itr.y`, `res.sol.itr.slx` and so on — dual values for the linear constraints

and many other fields of the *solution* structure (replace `itr` with `bas` or `int` for other solution types). Note that if the optimization failed then the `res.sol` field may not exist and attempting to access it will cause an error.

7.1.5 Source code example

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

Listing 7.1: Sample framework for checking optimization result.

```
function response(inputfile, solfile)


  cmd      = sprintf('read(%s)', inputfile)
  % Read the problem from file
  [r, res] = mosekopt(cmd)

  if strcmp( res.rcodestr , 'MSK_RES_OK')

      % Perform the optimization.
      [r,res] = mosekopt('minimize', res.prob);
      r
      res
      %Expected result: The solution status of the basic solution is optimal.
      if strcmp(res.rcodestr, 'MSK_RES_OK')

          solsta = strcat('MSK_SOL_STA_', res.sol.itr.solsta)

          if strcmp( solsta , 'MSK_SOL_STA_OPTIMAL')

              fprintf('An optimal basic solution is located.');

          elseif strcmp( solsta , 'MSK_SOL_STA_DUAL_INFEAS_CER')
              fprintf('Dual infeasibility certificate found.');

          elseif strcmp( solsta , 'MSK_SOL_STA_PRIM_INFEAS_CER')
            fprintf('Primal infeasibility certificate found.');

          elseif strcmp( solsta , 'MSK_SOL_STA_UNKNOWN')

              % The solutions status is unknown. The termination code
              % indicates why the optimizer terminated prematurely.

              fprintf('The solution status is unknown.');

              if ~strcmp(res.rcodestr, 'MSK_RES_OK' )

                  % A system failure e.g. out of space.
                  fprintf('  Response code: %s\n', res);

              else
```

(continues on next page)

(continued from previous page)

```
                %No system failure e.g. an iteration limit is reached.
                printf('  Termination code: %s\n', res);
            end

        else
          fprintf('An unexpected solution status is obtained.');
        end


    else
      fprintf('Could not obtain the solution status for the requested solution.');

  end

  fprintf('Return code: %d (0 means no error occurred.)\n',r);

end
```

7.2 Errors and exceptions

Response codes

The function *mosekopt* and its variants return a **response code** (and its human-readable description), informing if optimization was performed correctly, and if not, what error occurred. The expected response, indicating successful execution, is always *"MSK_RES_OK"*. Typical errors include:

- referencing a nonexisting variable (for example with too large index),
- incompatible dimensions of input data matrices,
- NaN in the input data,
- duplicate conic variable,
- error in the optimizer.

Some errors in data preprocessing, such as incorrect command or wrong parameter value will result in *mosekopt* exiting without assigning output; the error message will just be printed out. For this reason it may be a good idea to call *mosekopt* in a try-catch block. A full list of response codes, error, warning and termination codes can be found in the *API reference*. For example, the following code

```
prob.a = sparse(0,1);
prob.c = [NaN];
[r, res] = mosekopt('minimize', prob);
res
```

will produce as output:

```
res =

   rcode: 1470
    rmsg: 'The objective vector c contains an invalid value for variable '' (0).'
rcodestr: 'MSK_RES_ERR_NAN_IN_C'
```

Optimizer errors and warnings

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

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

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

7.3 Input/Output

7.3.1 Stream logging

By default the solver prints a log output analogous to the one produced by the command-line version of **MOSEK**. Logging may be turned off using the command `echo(0)`, for example:

```
[r, res] = mosekopt('minimize echo(0)', prob);
```

Log output may be redirected to a file using the command `log`, for example:

```
[r, res] = mosekopt('minimize log(fileName.txt)', prob);
```

Note that in recent versions of MATLAB the log is not displayed on screen until optimization is completed, which may be an inconvenience for longer tasks. The log written to a file does not have this issue.

Note also that leaving log output on can lead to a dramatic slowdown, visible especially on very small problems.

It is also possible to register a user-defined callback function that will receive and handle all log output, see the *callback* argument of *mosekopt*.

7.3.2 Log verbosity

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

- *MSK_IPAR_LOG*,
- *MSK_IPAR_LOG_INTPNT*,
- *MSK_IPAR_LOG_MIO*,
- *MSK_IPAR_LOG_CUT_SECOND_OPT*,
- *MSK_IPAR_LOG_SIM*, and
- *MSK_IPAR_LOG_SIM_MINOR*.

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

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

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

7.3.3 Saving a problem to a file

An optimization problem can be dumped to a file using the command `write`. The file format will be determined from the filename's extension. Supported formats are listed in Sec. 16 together with a table of problem types supported by each.

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

```
[r, res] = mosekopt('write(dump.opf)', prob);
```

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

```
[r, res] = mosekopt('write(dump.task.gz)', prob);
```

When using MATLAB-like functions the file name can be set using the `options` structure, for example:

```
opt.Write = 'problem.opf';
linprog(f,A,b,[],[],[],[],opt);
```

Some remarks:

- The problem is written to the file as it is represented in the underlying *optimizer task*, that is including any auxiliary variables introduced by the MATLAB-to-C interface, if applicable.
- Unnamed variables are given generic names. It is therefore recommended to use meaningful variable names if the problem file is meant to be human-readable.
- The `task` format is **MOSEK**'s native file format which contains all the problem data as well as solver settings.

7.3.4 Reading a problem from a file

A problem saved in any of the supported file formats can be read directly into a *`prob`* structure using the command `read`. Afterwards the problem can be optimized, modified, etc.

```
[r, res] = mosekopt('read(dump.opf.gz)');
prob = res.prob;
```

7.4 Setting solver parameters

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

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

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

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

Setting parameters

Each parameter is identified by a unique name and it can accept either integers, floating point values, symbolic strings or symbolic values. Parameters are set in the structure `param` and passed as a separate argument to *mosekopt*.

Some parameters can accept symbolic strings or symbolic values from a fixed set. The set of accepted values for every parameter is provided in the API reference.

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

Listing 7.2: Parameter setting example.

```
% Set log level (integer parameter)
param.MSK_IPAR_LOG = 1;
% Select interior-point optimizer... (integer parameter)
param.MSK_IPAR_OPTIMIZER = 'MSK_OPTIMIZER_INTPNT';
% ... without basis identification (integer parameter)
param.MSK_IPAR_INTPNT_BASIS = 'MSK_BI_NEVER';
% Set relative gap tolerance (double parameter)
param.MSK_DPAR_INTPNT_CO_TOL_REL_GAP = 1.0e-7;

% Use in mosekopt
[r,resp] = mosekopt('minimize', [], param);
```

7.5 Retrieving information items

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

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

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

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

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

Remark

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

Retrieving the values

Values of information items are only returned if the `info` command is used in *mosekopt*. They are available in the field `res.info`.

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

Listing 7.3: Information items example.

```
[~,res] = mosekopt('minimize info', []);

res.info.MSK_DINF_OPTIMIZER_TIME
res.info.MSK_IINF_INTPNT_ITER
```

7.6 Progress and data callback

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

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

7.6.1 Data callback

In the data callback **MOSEK** passes a callback code and values of all information items to a user-defined function. The callback function is called, in particular, at the beginning of each iteration of the interior-point optimizer. For the simplex optimizers *MSK_IPAR_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 attaching a structure *callback* as a parameter in *mosekopt*. This structure specifies a global callback function and can contain arbitrary user-defined data.

7.6.2 Progress callback

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

7.6.3 Working example: Data callback

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

Listing 7.4: An example of a data callback function.

```
function [r] = callback_handler(handle,where,info)

r = 0;   % r should always be assigned a value.

if handle.symbcon.MSK_CALLBACK_BEGIN_INTPNT==where
    disp(sprintf('Interior point optimizer started\n'));
end

if handle.symbcon.MSK_CALLBACK_END_INTPNT==where
  disp(sprintf('Interior-point optimizer terminated\n'));
  disp(sprintf('Interior-point primal obj.: %e\n', info.MSK_DINF_INTPNT_PRIMAL_OBJ));
  disp(sprintf('Iterations: %d\n', info.MSK_IINF_INTPNT_ITER));
end

if handle.symbcon.MSK_CALLBACK_NEW_INT_MIO==where
  disp(sprintf('New mixed-integer solution found\n'));
  disp(sprintf('Best objective.: %e\n', info.MSK_DINF_MIO_OBJ_BOUND));
```

(continues on next page)

(continued from previous page)

```
end

% Decide if to terminate the optimization
% Terminate when cputime > handle.maxtime
if info.MSK_DINF_INTPNT_TIME > handle.maxtime
  r = 1;
else
  r = 0;
end
```

Assuming that we have defined some problem `prob` the callback function is attached as follows:

Listing 7.5: Attaching the data callback function to the model.

```
% Define user defined handle.
[r,res]             = mosekopt('echo(0) symbcon');
data.maxtime        = 100.0;
data.symbcon        = res.symbcon;

callback.iter       = 'callback_handler';        % Defined in callback_handler.m
callback.iterhandle = data;

% Perform the optimization.
[r,res] = mosekopt('minimize echo(0)',prob,[],callback);
```

Chapter 8

Debugging Tutorials

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

The major steps when debugging a model are always:

- Consult the log output. It is enabled by default, but if neccessary switch it on explicitly with:

```
[r, res] = mosekopt('minimize echo(10)', prob);
```

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

  ```
  [r, res] = mosekopt('write(dump.opf)', prob);
  ```

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

  ```
  [r, res] = mosekopt('write(dump.task.gz)', prob);
  ```

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

See the following sections for details.

8.1 Understanding optimizer log

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

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

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

8.1.1 Input data

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

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

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

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

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

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

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

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

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

- Wrong type of cone was used or it has wrong dimension.
- The bounds for variables or constraints are incorrect or incomplete.
- The model is otherwise incomplete.
- Suspicious values of coefficients.
- For various data sizes the model does not scale as expected.

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

Warnings and errors

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

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

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

8.1.2 Solution summary

The last item in the log is the solution summary.

Continuous problem

Optimal solution

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

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

It contains the following elements:

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

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

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

Solution status UNKNOWN

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

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

Note that:

- The primal and dual objective are very different.
- The dual solution has very large norm.
- There are considerable violations so the solution is likely far from feasible.

Follow the hints in Sec. 8.2 to resolve the issue.

Solution status UNKNOWN with a potentially useful solution

Solution status UNKNOWN does not necessarily mean that the solution is completely useless. It only means that the solver was unable to make any more progress due to numerical difficulties, and it was not able to reach the accuracy required by the termination criteria (see Sec. 13.3.2). Consider for instance:

```
Problem status  : UNKNOWN
Solution status : UNKNOWN
Primal.  obj: 3.4531019648e+04    nrm: 1e+05    Viol.  con: 7e-02    var: 0e+00    cones: 0e+00
Dual.    obj: 3.4529720645e+04    nrm: 8e+03    Viol.  con: 1e-04    var: 2e-04    cones: 0e+00
```

Such a solution may still be useful, and it is always up to the user to decide. It may be a good enough approximation of the optimal point. For example, the large constraint violation may be due to the fact that one constraint contained a huge coefficient.

Infeasibility certificate

A primal infeasibility certificate is stored in the dual variables:

```
Problem status  : PRIMAL_INFEASIBLE
Solution status : PRIMAL_INFEASIBLE_CER
Dual.    obj: 2.9238975853e+02    nrm: 6e+02    Viol.  con: 0e+00    var: 1e-11    cones: 0e+00
```

It is a Farkas-type certificate as described in Sec. 12.2.2. In particular, for a good certificate:

- The dual objective is positive for a minimization problem, negative for a maximization problem. Ideally it is well bounded away from zero.
- The norm is not too big and the violations are small (as for a solution).

If the model was not expected to be infeasible, the likely cause is an error in the problem formulation. Use the hints in Sec. 8.1.1 and Sec. 8.3 to locate the issue.

Just like a solution, the infeasibility certificate can be of better or worse quality. The infeasibility certificate above is very solid. However, there can be less clear-cut cases, such as for example:

```
Problem status  : PRIMAL_INFEASIBLE
Solution status : PRIMAL_INFEASIBLE_CER
Dual.    obj: 1.6378689238e-06    nrm: 6e+05    Viol.  con: 7e-03    var: 2e-04    cones: 0e+00
```

This infeasibility certificate is more dubious because the dual objective is positive, but barely so in comparison with the large violations. It also has rather large norm. This is more likely an indication that the problem is borderline between feasibility and infeasibility or simply ill-posed and sensitive to tiny variations in input data. See Sec. 8.3 and Sec. 8.2.

The same remarks apply to dual infeasibility (i.e. unboundedness) certificates. Here the primal objective should be negative a minimization problem and positive for a maximization problem.

8.1.3 Mixed-integer problem

Optimal integer solution

For a mixed-integer problem there is no dual solution and a typical optimal solution report will look as follows:

```
Problem status  : PRIMAL_FEASIBLE
Solution status : INTEGER_OPTIMAL
Primal.  obj: 6.0111122960e+06    nrm: 1e+03    Viol.  con: 2e-13    var: 2e-14    itg: 5e-15
```

The interpretation of all elements is as for a continuous problem. The additional field `itg` denotes the maximum violation of an integer variable from being an exact integer.

Feasible integer solution

If the solver found an integer solution but did not prove optimality, for instance because of a time limit, the solution status will be `PRIMAL_FEASIBLE`:

```
Problem status  : PRIMAL_FEASIBLE
Solution status : PRIMAL_FEASIBLE
Primal.  obj: 6.0114607792e+06    nrm: 1e+03    Viol.  con: 2e-13    var: 2e-13    itg: 4e-15
```

In this case it is valuable to go back to the optimizer summary to see how good the best solution is:

```
31       35       1        0        6.0114607792e+06     6.0078960892e+06     0.06        4.1

Objective of best integer solution : 6.011460779193e+06
Best objective bound               : 6.007896089225e+06
```

In this case the best integer solution found has objective value `6.011460779193e+06`, the best proved lower bound is `6.007896089225e+06` and so the solution is guaranteed to be within 0.06% from optimum. The same data can be obtained as information items through an API. See also Sec. 13.4 for more details.

Infeasible problem

If the problem is declared infeasible the summary is simply

```
Problem status  : PRIMAL_INFEASIBLE
Solution status : UNKNOWN
Primal.  obj: 0.0000000000e+00    nrm: 0e+00    Viol.  con: 0e+00    var: 0e+00    itg: 0e+00
```

If infeasibility was not expected, consult Sec. 8.3.

8.2 Addressing numerical issues

The suggestions in this section should help diagnose and solve issues with numerical instability, in particular `UNKNOWN` solution status or solutions with large violations. Since numerically stable models tend to solve faster, following these hints can also dramatically shorten solution times.

We always recommend that issues of this kind are addressed by reformulating or rescaling the model, since it is the modeler who has the best insight into the structure of the problem and can fix the cause of the issue.

8.2.1 Formulating problems

Scaling

Make sure that all the data in the problem are of comparable orders of magnitude. This applies especially to the linear constraint matrix. Use Sec. 8.1.1 if necessary. For example a report such as

```
|A|                nnz: 597023       min=1.17e-6       max=2.21e+5
```

means that the ratio of largest to smallest elements in `A` is 10^{11}. In this case the user should rescale or reformulate the model to avoid such spread which makes it difficult for **MOSEK** to scale the problem internally. In many cases it may be possible to change the units, i.e. express the model in terms of rescaled variables (for instance work with millions of dollars instead of dollars, etc.).

Similarly, if the objective contains very different coefficients, say

$$\text{maximize } 10^{10}x + y$$

then it is likely to lead to inaccuracies. The objective will be dominated by the contribution from x and y will become insignificant.

Removing huge bounds

Never use a very large number as replacement for ∞. Instead define the variable or constraint as unbounded from below/above. Similarly, avoid artificial huge bounds if you expect they will not become tight in the optimal solution.

Avoiding linear dependencies

As much as possible try to avoid linear dependencies and near-linear dependencies in the model. See Example 8.3.

Avoiding ill-posedness

Avoid continuous models which are ill-posed: the solution space is degenerate, for example consists of a single point (technically, the Slater condition is not satisfied). In general, this refers to problems which are borderline between feasible and infeasible. See Example 8.1.

Scaling the expected solution

Try to formulate the problem in such a way that the expected solution (both primal and dual) is not very large. Consult the solution summary Sec. 8.1.2 to check the objective values or solution norms.

8.2.2 Further suggestions

Here are other simple suggestions that can help locate the cause of the issues. They can also be used as hints for how to tune the optimizer if fixing the root causes of the issue is not possible.

- Remove the objective and solve the feasibility problem. This can reveal issues with the objective.
- Change the objective or change the objective sense from minimization to maximization (if applicable). If the two objective values are almost identical, this may indicate that the feasible set is very small, possibly degenerate.
- Perturb the data, for instance bounds, very slightly, and compare the results.
- For linear problems: solve the problem using a different optimizer by setting the parameter *MSK_IPAR_OPTIMIZER* and compare the results.
- Force the optimizer to solve the primal/dual versions of the problem by setting the parameter *MSK_IPAR_INTPNT_SOLVE_FORM* or *MSK_IPAR_SIM_SOLVE_FORM*. **MOSEK** has a heuristic to decide whether to dualize, but for some problems the guess is wrong an explicit choice may give better results.
- Solve the problem without presolve or some of its parts by setting the parameter *MSK_IPAR_PRESOLVE_USE*, see Sec. 13.1.
- Use different numbers of threads (*MSK_IPAR_NUM_THREADS*) and compare the results. Very different results indicate numerical issues resulting from round-off errors.

If the problem was dumped to a file, experimenting with various parameters is facilitated with the **MOSEK** Command Line Tool or **MOSEK** Python Console Sec. 8.4.

8.2.3 Typical pitfalls

Example 8.1 (Ill-posedness). A toy example of this situation is the feasibility problem

$$(x-1)^2 \leq 1, \ (x+1)^2 \leq 1$$

whose only solution is $x = 0$ and moreover replacing any 1 on the right hand side by $1 - \varepsilon$ makes the problem infeasible and replacing it by $1 + \varepsilon$ yields a problem whose solution set is an interval (fully-dimensional). This is an example of ill-posedness.

Example 8.2 (Huge solution)**.** If the norm of the expected solution is very large it may lead to numerical issues or infeasibility. For example the problem

$$(10^{-4}, x, 10^3) \in \mathcal{Q}_r^3$$

may be declared infeasible because the expected solution must satisfy $x \geq 5 \cdot 10^9$.

Example 8.3 (Near linear dependency)**.** Consider the following problem:

$$\begin{array}{lrcrcrcrcl}
\text{minimize} & & & & & & & & & \\
\text{subject to} & & x_1 & + & x_2 & & & & = & 1, \\
 & & & & & & x_3 & + \; x_4 & = & 1, \\
 & - & x_1 & & & - & x_3 & & = & -1+\varepsilon, \\
 & & & - & x_2 & & & - \; x_4 & = & -1, \\
 & & x_1, & & x_2, & & x_3, & x_4 & \geq & 0.
\end{array}$$

If we add the equalities together we obtain:

$$0 = \varepsilon$$

which is infeasible for any $\varepsilon \neq 0$. Here infeasibility is caused by a linear dependency in the constraint matrix coupled with a precision error represented by the ε. Indeed if a problem contains linear dependencies then the problem is either infeasible or contains redundant constraints. In the above case any of the equality constraints can be removed while not changing the set of feasible solutions. To summarize linear dependencies in the constraints can give rise to infeasible problems and therefore it is better to avoid them.

Example 8.4 (Presolving very tight bounds)**.** Next consider the problem

$$\begin{array}{lrcr}
\text{minimize} & & & \\
\text{subject to} & x_1 - 0.01x_2 & = & 0, \\
 & x_2 - 0.01x_3 & = & 0, \\
 & x_3 - 0.01x_4 & = & 0, \\
 & x_1 & \geq & -10^{-9}, \\
 & x_1 & \leq & 10^{-9}, \\
 & x_4 & \geq & 10^{-4}.
\end{array}$$

Now the **MOSEK** presolve will, for the sake of efficiency, fix variables (and constraints) that have tight bounds where tightness is controlled by the parameter *MSK_DPAR_PRESOLVE_TOL_X*. Since the bounds

$$-10^{-9} \leq x_1 \leq 10^{-9}$$

are tight, presolve will set $x_1 = 0$. It easy to see that this implies $x_4 = 0$, which leads to the incorrect conclusion that the problem is infeasible. However a tiny change of the value 10^{-9} makes the problem feasible. In general it is recommended to avoid ill-posed problems, but if that is not possible then one solution is to reduce parameters such as *MSK_DPAR_PRESOLVE_TOL_X* to say 10^{-10}. This will at least make sure that presolve does not make the undesired reduction.

8.3 Debugging infeasibility

This section contains hints for debugging problems that are unexpectedly infeasible. It is always a good idea to remove the objective, i.e. only solve a feasibility problem when debugging such issues.

8.3.1 Numerical issues

Infeasible problem status may be just an artifact of numerical issues appearing when the problem is badly-scaled, barely feasible or otherwise ill-conditioned so that it is unstable under small perturbations of the data or round-off errors. This may be visible in the solution summary if the infeasibility certificate has poor quality. See Sec. 8.1.2 for how to diagnose that and Sec. 8.2 for possible hints. Sec. 8.2.3 contains examples of situations which may lead to infeasibility for numerical reasons.

We refer to Sec. 8.2 for further information on dealing with those sort of issues. For the rest of this section we concentrate on the case when the solution summary leaves little doubt that the problem solved by the optimizer actually is infeasible.

8.3.2 Locating primal infeasibility

As an example of a primal infeasible problem consider minimizing the cost of transportation between a number of production plants and stores: Each plant produces a fixed number of goods, and each store has a fixed demand that must be met. Supply, demand and cost of transportation per unit are given in Fig. 8.1.

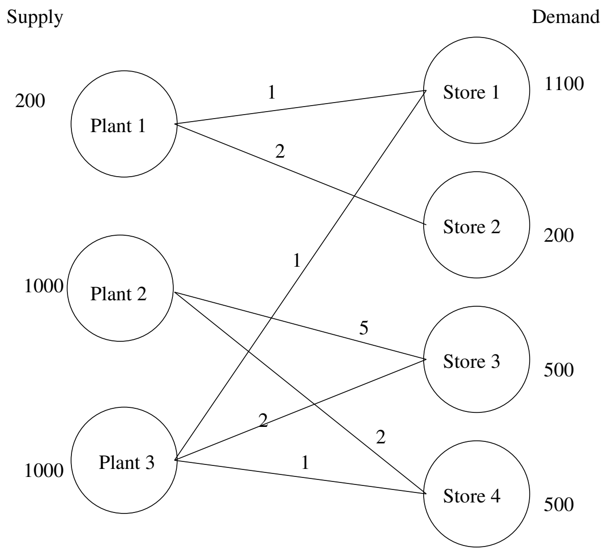

Fig. 8.1: Supply, demand and cost of transportation.

The problem represented in Fig. 8.1 is infeasible, since the total demand

$$2300 = 1100 + 200 + 500 + 500$$

exceeds the total supply

$$2200 = 200 + 1000 + 1000$$

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

$$\begin{array}{llccccccccccccccl}
\text{minimize} & & x_{11} & + & 2x_{12} & + & 5x_{23} & + & 2x_{24} & + & x_{31} & + & 2x_{33} & + & x_{34} & & \\
\text{subject to} & s_0: & x_{11} & + & x_{12} & & & & & & & & & & & \leq & 200, \\
 & s_1: & & & & & x_{23} & + & x_{24} & & & & & & & \leq & 1000, \\
 & s_2: & & & & & & & & & x_{31} & + & x_{33} & + & x_{34} & \leq & 1000, \\
 & d_1: & x_{11} & & & & & & & + & x_{31} & & & & & = & 1100, \\
 & d_2: & & & x_{12} & & & & & & & & & & & = & 200, \\
 & d_3: & & & & & x_{23} & + & & & & & x_{33} & & & = & 500, \\
 & d_4: & & & & & & & x_{24} & + & & & & & x_{34} & = & 500, \\
 & & & & & & & & & & & & & & x_{ij} & \geq & 0.
\end{array} \tag{8.1}$$

Solving problem (8.1) using **MOSEK** will result in an infeasibility status. The infeasibility certificate is contained in the dual variables an can be accessed from an API. The variables and constraints with nonzero solution values form an infeasible subproblem, which frequently is very small. See Sec. 12.1.2 or Sec. 12.2.2 for detailed specifications of infeasibility certificates.

A short infeasibility report can also be printed to the log stream. It can be turned on by setting the parameter *MSK_IPAR_INFEAS_REPORT_AUTO* to *"MSK_ON"*. This causes **MOSEK** to print a report on variables and constraints which are involved in infeasibility in the above sense, i.e. have nonzero values in the certificate. The parameter *MSK_IPAR_INFEAS_REPORT_LEVEL* controls the amount of information presented in the infeasibility report. The default value is 1. For the above example the report is

```
MOSEK PRIMAL INFEASIBILITY REPORT.

Problem status: The problem is primal infeasible

The following constraints are involved in the primal infeasibility.

Index     Name       Lower bound       Upper bound       Dual lower        Dual upper
0         s0         NONE              2.000000e+002     0.000000e+000     1.000000e+000
2         s2         NONE              1.000000e+003     0.000000e+000     1.000000e+000
3         d1         1.100000e+003     1.100000e+003     1.000000e+000     0.000000e+000
4         d2         2.000000e+002     2.000000e+002     1.000000e+000     0.000000e+000

The following bound constraints are involved in the infeasibility.

Index     Name       Lower bound       Upper bound       Dual lower        Dual upper
8         x33        0.000000e+000     NONE              1.000000e+000     0.000000e+000
10        x34        0.000000e+000     NONE              1.000000e+000     0.000000e+000
```

The infeasibility report is divided into two sections corresponding to constraints and variables. It is a selection of those lines from the problem solution which are important in understanding primal infeasibility. In this case the constraints `s0`, `s2`, `d1`, `d2` and variables `x33`, `x34` are of importance because of nonzero dual values. The columns `Dual lower` and `Dual upper` contain the values of dual variables s_l^c, s_u^c, s_l^x and s_u^x in the primal infeasibility certificate (see Sec. 12.1.2).

In our example the certificate means that an appropriate linear combination of constraints `s0`, `s1` with coefficient $s_u^c = 1$, constraints `d1` and `d2` with coefficient $s_u^c - s_l^c = 0 - 1 = -1$ and lower bounds on `x33` and `x34` with coefficient $-s_l^x = -1$ gives a contradiction. Indeed, the combination of the four involved constraints is $x_{33} + x_{34} \leq -100$ (as indicated in the introduction, the difference between supply and demand).

It is also possible to extract the infeasible subproblem with the command-line tool. For an infeasible problem called `infeas.lp` the command:

```
mosek -d MSK_IPAR_INFEAS_REPORT_AUTO MSK_ON infeas.lp -info rinfeas.lp
```

will produce the file `rinfeas.bas.inf.lp` which contains the infeasible subproblem. Because of its size it may be easier to work with than the original problem file.

Returning to the transportation example, we discover that removing the fifth constraint $x_{12} = 200$ makes the problem feasible. Almost all undesired infeasibilities should be fixable at the modeling stage.

8.3.3 Locating dual infeasibility

A problem may also be *dual infeasible*. In this case the primal problem is usually unbounded, meaning that feasible solutions exists such that the objective tends towards infinity. For example, consider the problem

$$
\begin{array}{ll}
\text{maximize} & 200y_1 + 1000y_2 + 1000y_3 + 1100y_4 + 200y_5 + 500y_6 + 500y_7 \\
\text{subject to} & y_1 + y_4 \leq 1,\ y_1 + y_5 \leq 2,\ y_2 + y_6 \leq 5,\ y_2 + y_7 \leq 2, \\
& y_3 + y_4 \leq 1,\ y_3 + y_6 \leq 2,\ y_3 + y_7 \leq 1 \\
& y_1, y_2, y_3 \leq 0
\end{array}
$$

which is dual to (8.1) (and therefore is dual infeasible). The dual infeasibility report may look as follows:

```
MOSEK DUAL INFEASIBILITY REPORT.

Problem status: The problem is dual infeasible

The following constraints are involved in the infeasibility.

Index     Name              Activity         Objective        Lower bound      Upper bound
5         x33               -1.000000e+00                     NONE             2.000000e+00
6         x34               -1.000000e+00                     NONE             1.000000e+00

The following variables are involved in the infeasibility.

Index     Name              Activity         Objective        Lower bound      Upper bound
0         y1                -1.000000e+00    2.000000e+02     NONE             0.000000e+00
2         y3                -1.000000e+00    1.000000e+03     NONE             0.000000e+00
3         y4                1.000000e+00     1.100000e+03     NONE             NONE
4         y5                1.000000e+00     2.000000e+02     NONE             NONE

Interior-point solution summary
  Problem status  : DUAL_INFEASIBLE
  Solution status : DUAL_INFEASIBLE_CER
  Primal.  obj: 1.0000000000e+02    nrm: 1e+00    Viol.  con: 0e+00    var: 0e+00
```

In the report we see that the variables `y1`, `y3`, `y4`, `y5` and two constraints contribute to infeasibility with non-zero values in the `Activity` column. Therefore

$$(y_1, \ldots, y_7) = (-1, 0, -1, 1, 1, 0, 0)$$

is the dual infeasibility certificate as in Sec. 12.1.2. This just means, that along the ray

$$(0, 0, 0, 0, 0, 0, 0) + t(y_1, \ldots, y_7) = (-t, 0, -t, t, t, 0, 0), \ \ t > 0,$$

which belongs to the feasible set, the objective value $100t$ can be arbitrarily large, i.e. the problem is unbounded.

In the example problem we could

- Add a lower bound on `y3`. This will directly invalidate the certificate of dual infeasibility.
- Increase the objective coefficient of `y3`. Changing the coefficients sufficiently will invalidate the inequality $c^T y^* > 0$ and thus the certificate.

8.3.4 Suggestions

Primal infeasibility

When trying to understand what causes the unexpected primal infeasible status use the following hints:

- Remove the objective function. This does not change the infeasibility status but simplifies the problem, eliminating any possibility of issues related to the objective function.
- Remove cones, semidefinite variables and integer constraints. Solve only the linear part of the problem. Typical simple modeling errors will lead to infeasibility already at this stage.
- Consider whether your problem has some obvious necessary conditions for feasibility and examine if these are satisfied, e.g. total supply should be greater than or equal to total demand.
- Verify that coefficients and bounds are reasonably sized in your problem.
- See if there are any obvious contradictions, for instance a variable is bounded both in the variables and constraints section, and the bounds are contradictory.
- Consider replacing suspicious equality constraints by inequalities. For instance, instead of $x_{12} = 200$ see what happens for $x_{12} \geq 200$ or $x_{12} \leq 200$.

- Relax bounds of the suspicious constraints or variables.
- For integer problems, remove integrality constraints on some/all variables and see if the problem solves.
- Form an **elastic model**: allow to violate constraints at a cost. Introduce slack variables and add them to the objective as penalty. For instance, suppose we have a constraint

$$\begin{array}{ll}\text{minimize} & c^T x,\\ \text{subject to} & a^T x \leq b.\end{array}$$

  which might be causing infeasibility. Then create a new variable y and form the problem which contains:

$$\begin{array}{ll}\text{minimize} & c^T x + y,\\ \text{subject to} & a^T x \leq b + y.\end{array}$$

  Solving this problem will reveal by how much the constraint needs to be relaxed in order to become feasible. This is equivalent to inspecting the infeasibility certificate but may be more intuitive.
- If you think you have a feasible solution or its part, fix all or some of the variables to those values. Presolve will propagate them through the model and potentially reveal more localized sources of infeasibility.
- Dump the problem in OPF or LP format and verify that the problem that was passed to the optimizer corresponds to the problem expressed in the high-level modeling language, if any such was used.

Dual infeasibility

When trying to understand what causes the unexpected dual infeasible status use the following hints:

- Verify that the objective coefficients are reasonably sized.
- Check if no bounds and constraints are missing, for example if all variables that should be nonnegative have been declared as such etc.
- Strengthen bounds of the suspicious constraints or variables.
- Form an series of models with decreasing bounds on the objective, that is, instead of objective

$$\text{minimize } c^T x$$

  solve the problem with an additional constraint such as

$$c^T x = -10^5$$

  and inspect the solution to figure out the mechanism behind arbitrarily decreasing objective values. This is equivalent to inspecting the infeasibility certificate but may be more intuitive.
- Dump the problem in OPF or LP format and verify that the problem that was passed to the optimizer corresponds to the problem expressed in the high-level modeling language, if any such was used.

Please note that modifying the problem to invalidate the reported certificate does *not* imply that the problem becomes feasible — the reason for infeasibility may simply *move*, resulting a problem that is still infeasible, but for a different reason. More often, the reported certificate can be used to give a hint about errors or inconsistencies in the model that produced the problem.

8.4 Python Console

The **MOSEK** Python Console is an alternative to the **MOSEK** Command Line Tool. It can be used for interactive loading, solving and debugging optimization problems stored in files, for example **MOSEK** task files. It facilitates debugging techniques described in Sec. 8.

8.4.1 Usage

The tool requires Python 2 or 3. The **MOSEK** interface for Python must be installed following the installation instructions for Python API or Python Fusion API. In the basic case it should be sufficient to execute the script

```
python setup.py install --user
```

in the directory containing the **MOSEK** Python module.

The Python Console is contained in the file `mosekconsole.py` in the folder with **MOSEK** binaries. It can be copied to an arbitrary location. The file is also available for `download here (mosekconsole.py)`.

To run the console in interactive mode use

```
python mosekconsole.py
```

To run the console in batch mode provide a semicolon-separated list of commands as the second argument of the script, for example:

```
python mosekconsole.py "read data.task.gz; solve form=dual; writesol data"
```

The script is written using the **MOSEK** Python API and can be extended by the user if more specific functionality is required. We refer to the documentation of the Python API.

8.4.2 Examples

To read a problem from `data.task.gz`, solve it, and write solutions to `data.sol`, `data.bas` or `data.itg`:

```
read data.task.gz; solve; writesol data
```

To convert between file formats:

```
read data.task.gz; write data.mps
```

To set a parameter before solving:

```
read data.task.gz; param INTPNT_CO_TOL_DFEAS 1e-9; solve"
```

To list parameter values related to the mixed-integer optimizer in the task file:

```
read data.task.gz; param MIO
```

To print a summary of problem structure:

```
read data.task.gz; anapro
```

To solve a problem forcing the dual and switching off presolve:

```
read data.task.gz; solve form=dual presolve=no
```

To write an infeasible subproblem to a file for debugging purposes:

```
read data.task.gz; solve; infsub; write inf.opf
```

8.4.3 Full list of commands

Below is a brief description of all the available commands. Detailed information about a specific command `cmd` and its options can be obtained with

```
help cmd
```

Table 8.1: List of commands of the MOSEK Python Console.

Command	Description
`help [command]`	Print list of commands or info about a specific command
`log filename`	Save the session to a file
`intro`	Print MOSEK splashscreen
`testlic`	Test the license system
`read filename`	Load problem from file
`reread`	Reload last problem file
`solve [options]`	Solve current problem
`write filename`	Write current problem to file
`param [name [value]]`	Set a parameter or get parameter values
`info [name]`	Get an information item
`anapro`	Analyze problem data
`hist`	Plot a histogram of problem data
`anasol`	Analyze solutions
`removeitg`	Remove integrality constraints
`infsub`	Replace current problem with its infeasible subproblem
`writesol basename`	Write solution(s) to file(s) with given basename
`delsol`	Remove all solutions from the task
`exit`	Leave

Chapter 9

Advanced Numerical Tutorials

9.1 Converting a quadratically constrained problem to conic form

MOSEK employs the following form of quadratic problems:

$$\begin{array}{lccccl} \text{minimize} & & & \frac{1}{2}x^TQ^ox + c^Tx + c^f & & \\ \text{subject to} & l_k^c & \leq & \frac{1}{2}x^TQ^kx + \sum_{j=0}^{n-1} a_{k,j}x_j & \leq & u_k^c, \quad k = 0,\ldots,m-1, \\ & l_j^x & \leq & x_j & \leq & u_j^x, \quad j = 0,\ldots,n-1. \end{array} \tag{9.1}$$

A conic quadratic constraint has the form

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

in its most basic form where

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

A quadratic problem such as (9.1), if convex, can be reformulated in conic form. This is in fact the reformulation **MOSEK** performs internally. It has many advantages:

- elegant duality theory for conic problems,
- reporting accurate dual information for quadratic inequalities is hard and/or computational expensive,
- it certifies that the original quadratic problem is indeed convex,
- modeling directly in conic form usually leads to a better model *[And13]* i.e. a faster solution time and better numerical properties.

In addition, there are more types of conic constraints that can be combined with a quadratic cone, for example semidefinite cones.

MOSEK offers a function that performs the conversion from quadratic to conic quadratic form explicitly. Note that the reformulation is not unique. The approach followed by **MOSEK** is to introduce additional variables, linear constraints and quadratic cones to obtain a larger but equivalent problem in which the original variables are preserved.

In particular:

- all variables and constraints are kept in the problem,
- each quadratic constraint and quadratic terms in the objective generate one rotated quadratic cone,
- each quadratic constraint will contain no coefficients and upper/lower bounds will be set to $\infty, -\infty$ respectively.

This allows the user to recover the original variable and constraint values, as well as their dual values, with no conversion or additional effort.

9.1.1 Quadratic Constraint Reformulation

Let us assume we want to convert the following quadratic constraint

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

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

$$\frac{1}{2}x^T Q x + \sum_{j=0}^{n-1} a_j^T x_j \le u. \tag{9.2}$$

Introducing an additional variable w such that

$$w = u - \sum_{j=0}^{n-1} a_j^T x_j \tag{9.3}$$

we obtain the equivalent form

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

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

$$Q = FF^T \tag{9.4}$$

and therefore we can write

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

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

$$\begin{array}{ll} (w, z, y) & \in \mathcal{Q}_r, \\ z = 1 & \\ y = Fx & \\ w = u - a^T x. & \end{array} \tag{9.5}$$

Summarizing, for each quadratic constraint involving t variables, **MOSEK** introduces

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

A quadratic term in the objective is reformulated in a similar fashion. We refer to *[And13]* for a more thorough discussion.

Example

Next we consider a simple problem with quadratic objective function:

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}(13x_0^2 + 17x_1^2 + 12x_2^2 + 24x_0x_1 + 12x_1x_2 - 4x_0x_2) - 22x_0 - 14.5x_1 + 12x_2 + 1 \\ \text{subject to} & -1 \le x_0, x_1, x_2 \le 1 \end{array}$$

We can specify it in the human-readable OPF format.

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

[variables]
x0 x1 x2
[/variables]

[objective min]
0.5 (13 x0^2  + 17 x1^2 + 12 x2^2 + 24 x0 * x1 + 12 x1 * x2 - 4 x0 * x2 ) - 22 x0 - 14.5 x1 +␣
↪12 x2 + 1
[/objective]

[bounds]
[b] -1 <= * <= 1 [/b]
[/bounds]
```

The objective function is convex, the minimum is attained for $x^\star = (1, 0.5, -1)$. The conversion will introduce first a variable x_3 in the objective function such that $x_3 \geq 1/2x^TQx$ and then convert the latter directly in conic form. The converted problem follows:

$$
\begin{array}{lr}
\text{minimize} & -22x_0 - 14.5x_1 + 12x_2 + x_3 + 1 \\
\text{subject to} & 3.61x_0 + 3.33x_1 - 0.55x_2 - x_6 = 0 \\
 & +2.29x_1 + 3.42x_2 - x_7 = 0 \\
 & 0.81x_1 - x_8 = 0 \\
 & -x_3 + x_4 = 0 \\
 & x_5 = 1 \\
 & (x_4, x_5, x_6, x_7, x_8) \in \mathcal{Q}_\nabla \\
 & -1 \leq x_0, x_1, x_2 \leq 1
\end{array}
$$

We obtain the reformulation as follows:

```
% prob is a quadratic problem
[r, res] = mosekopt('toconic prob', prob)
probConic = res.prob
mosekopt('write(conic.opf)', probConic)
```

and the output is:

```
[comment]
   Written by MOSEK version 8.1.0.19
   Date 21-08-17
   Time 10:53:36
[/comment]

[hints]
  [hint NUMVAR] 9 [/hint]
  [hint NUMCON] 4 [/hint]
  [hint NUMANZ] 11 [/hint]
  [hint NUMQNZ] 0 [/hint]
  [hint NUMCONE] 1 [/hint]
[/hints]

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

[objective minimize]
   - 2.2e+01 x0000_x0 - 1.45e+01 x0001_x1 + 1.2e+01 x0002_x2 + x0003
   + 1e+00
```

(continues on next page)

(continued from previous page)

```
[/objective]

[constraints]
  [con c0000]  3.605551275463989e+00 x0000_x0 - 5.547001962252291e-01 x0002_x2 + 3.
↪328201177351375e+00 x0001_x1 - x0006 = 0e+00 [/con]
  [con c0001]  3.419401657060442e+00 x0002_x2 + 2.294598480395823e+00 x0001_x1 - x0007 = 0e+00␣
↪[/con]
  [con c0002]  8.111071056538127e-01 x0001_x1 - x0008 = 0e+00 [/con]
  [con c0003]  - x0003 + x0004 = 0e+00 [/con]
[/constraints]

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

We can clearly see that constraints c0000, c0001 and c0002 represent the original linear constraints as in (9.4), while c0003 corresponds to (9.3). The cone roots are x0005 and x0004.

9.2 Advanced hot-start

In practice it frequently occurs that when an optimization problem has been solved, then the same problem slightly modified should be reoptimized. Moreover, if it is just a small the modification, it can be expected that the optimal solution to the original problem is a good approximation to the modified problem. Therefore, it should be efficient to start the optimization of the modified problem from the previous optimal solution.

Currently, the interior-point optimizer in **MOSEK** cannot take advantage of a previous optimal solution, however, the simplex optimizer can exploit any basic solution.

We work with the simple linear problem:

$$\begin{array}{ll} \text{minimize} & x_1 + 2x_2 \\ \text{subject to} & 4 \leq x_1 + x_3 \leq 6, \\ & 1 \leq x_1 + x_2, \\ & 0 \leq x_1, x_2, x_3. \end{array}$$

9.2.1 Initial hot-start

A quick inspection of the problem indicates that $(x_1, x_3) = (1, 3)$ is an optimal solution. Hence, it seems to be a good idea to let the initial basis consist of x_1 and x_3 and all the other variables be at their lower bounds. This idea is used in the example code:

Listing 9.1: Passing the full basic solution.

```
% Specify an initial basic solution.
bas.skc      = ['LL';'LL'];
bas.skx      = ['BS';'LL';'BS'];
bas.xc       = [4 1]';
bas.xx       = [1 3 0]';

prob.sol.bas = bas;

% Specify the problem data.
prob.c       = [ 1 2 0]';
subi         = [1 2 2 1];
subj         = [1 1 2 3];
valij        = [1.0 1.0 1.0 1.0];
```

(continues on next page)

(continued from previous page)

```
prob.a       = sparse(subi,subj,valij);
prob.blc     = [4.0 1.0]';
prob.buc     = [6.0 inf]';
prob.blx     = sparse(3,1);
prob.bux     = [];

% Use the primal simplex optimizer.
param.MSK_IPAR_OPTIMIZER = 'MSK_OPTIMIZER_PRIMAL_SIMPLEX';
[r,res] = mosekopt('minimize',prob,param)
```

Comments:

- In the example the dual solution is not defined. This is acceptable because the primal simplex optimizer is used for the reoptimization and it does not exploit a dual solution. Otherwise it will be important that a good dual solution is specified.
- The status keys `bas.skc` and `bas.skx` must contain only the entries `BS`, `EQ`, `LL`, `UL`, `SB`. Moreover, e.g. `EQ` must be specified only for a fixed constraint or variable. `LL` and `UL` can be used only for a variable that has a finite lower or upper bound respectively. For an explanation of status keys see *stakey*.
- The number of constraints and variables defined to be basic must correspond exactly to the number of constraints.

9.2.2 Adding a new variable

Next, assume we modify the problem by adding a new variable:

$$\begin{array}{ll} \mbox{minimize} & x_1 + 2x_2 - x_4 \\ \mbox{subject to} & 4 \leq x_1 + x_3 + x_4 \leq 6, \\ & 1 \leq x_1 + x_2, \\ & 0 \leq x_1, x_2, x_3, x_4. \end{array}$$

In continuation of the previous example this problem can be solved as follows, using the full previous basic solution in hot-start:

Listing 9.2: Hot-start when adding a new variable.

```
prob.c       = [prob.c;-1.0];
prob.a       = [prob.a,sparse([1.0 0.0]')];
prob.blx     = sparse(4,1);

% Reuse the old optimal basic solution.
bas          = res.sol.bas;

% Add to the status key.
bas.skx      = [res.sol.bas.skx;'LL'];

% The new variable is at it lower bound.
bas.xx       = [res.sol.bas.xx;0.0];
bas.slx      = [res.sol.bas.slx;0.0];
bas.sux      = [res.sol.bas.sux;0.0];

prob.sol.bas = bas;

[rcode,res]  = mosekopt('minimize',prob,param);

% The new primal optimal solution
res.sol.bas.xx'
```

9.2.3 Fixing a variable

In e.g. branch-and-bound methods for integer programming problems it is necessary to reoptimize the problem after a variable has been fixed to a value. This can easily be achieved as follows:

Listing 9.3: Hot-start with a fixed variable.

```
prob.blx(4)  = 1;
prob.bux     = [inf inf inf 1]';

% Reuse the basis.
prob.sol.bas = res.sol.bas;

[rcode,res]  = mosekopt('minimize',prob,param);

% Display the optimal solution.
res.sol.bas.xx'
```

9.2.4 Adding a new constraint

Now assume that the constraint

$$x_1 + x_2 \geq 2$$

should be added to the problem and the problem should be reoptimized. The following example demonstrates how to do this.

Listing 9.4: Hot-start when adding a new constraint.

```
% Modify the problem.
prob.a       = [prob.a;sparse([1.0 1.0 0.0 0.0])];
prob.blc     = [prob.blc;2.0];
prob.buc     = [prob.buc;inf];

% Obtain the previous optimal basis.
bas          = res.sol.bas;

% Set the solution to the modified problem.
bas.skc      = [bas.skc;'BS'];
bas.xc       = [bas.xc;bas.xx(1)+bas.xx(2)];
bas.y        = [bas.y;0.0];
bas.slc      = [bas.slc;0.0];
bas.suc      = [bas.suc;0.0];

% Reuse the basis.
prob.sol.bas = bas;

% Reoptimize.
[rcode,res]  = mosekopt('minimize',prob,param);

res.sol.bas.xx'
```

Please note that the slack variable corresponding to the new constraint is declared basic. This implies that the new basis is nonsingular and can be reused.

9.2.5 Removing a constraint

We can remove a constraint in two ways:

- Set the bounds for the constraint to $\pm\infty$ as appropriate.
- Remove the corresponding row from `prob.a` and other parts of the data and update the basis.

In the following example we use the latter approach to again remove the constraint $x_1 + x_2 \geq 2$.

Listing 9.5: Hot-start when removing a constraint.

```
% Modify the problem.
prob.a       = prob.a(1:end-1,:);
prob.blc     = prob.blc(1:end-1);
prob.buc     = prob.buc(1:end-1);

% Obtain the previous optimal basis.
bas          = res.sol.bas;

% Set the solution to the modified problem.
bas.skc      = bas.skc(1:end-1,:);
bas.xc       = bas.xc(1:end-1);
bas.y        = bas.y(1:end-1);
bas.slc      = bas.slc(1:end-1);
bas.suc      = bas.suc(1:end-1);

% Reuse the basis.
prob.sol.bas = bas;

% Reoptimize.
[rcode,res]  = mosekopt('minimize',prob,param);

res.sol.bas.xx'
```

9.2.6 Removing a variable

Similarly we can remove a variable in two ways:

- Fix the variable to zero.
- Remove the corresponding column from `prob.a` and other parts of the data and update the basis.

The following example uses the latter approach to remove x_4.

Listing 9.6: Hot-start when removing a constraint.

```
% Modify the problem.
prob.c       = prob.c(1:end-1);
prob.a       = prob.a(:,1:end-1);
prob.blx     = prob.blx(1:end-1);
prob.bux     = prob.bux(1:end-1);

% Obtain the previous optimal basis.
bas          = res.sol.bas;

% Set the solution to the modified problem.
bas.xx       = bas.xx(1:end-1);
bas.skx      = bas.skx(1:end-1,:);
bas.slx      = bas.slx(1:end-1);
bas.sux      = bas.sux(1:end-1);

% Reuse the basis.
prob.sol.bas = bas;

% Reoptimize.
[rcode,res]  = mosekopt('minimize',prob,param);

res.sol.bas.xx'
```

Chapter 10

Technical guidelines

This section contains some more in-depth technical guidelines for Optimization Toolbox for MATLAB, not strictly necessary for basic use of **MOSEK**.

10.1 Integration with MATLAB

The mosekopt MEX file

The central part of Optimization Toolbox for MATLAB is the *mosekopt* MEX file. It provides an interface to **MOSEK** that is employed by all the other functions provided in the toolbox. Therefore, we recommend to *mosekopt* function if possible because that give rise to the least overhead and provides the maximum of features.

Compatibility with the MATLAB Optimization Toolbox

For compatibility with the MATLAB Optimization Toolbox, **MOSEK** provides the following functions:

- *linprog*: Solves linear optimization problems.
- *intlinprog*: Solves a linear optimization problem with integer constrained variables.
- *quadprog*: Solves quadratic optimization problems.
- *lsqlin*: Minimizes a least-squares objective with linear constraints.
- *lsqnonneg*: Minimizes a least-squares objective with nonnegativity constraints.
- *mskoptimget*: Getting an `options` structure for MATLAB compatible functions.
- *mskoptimset*: Setting up an `options` structure for MATLAB compatible functions.

These functions are described in detail in Sec. 15.2. The functions *mskoptimget* and *mskoptimset* are not fully compatible with the MATLAB counterparts, `optimget` and `optimset`, so the **MOSEK** versions should only be used in conjunction with the **MOSEK** implementations of *linprog*, etc., and similarly `optimget` should be used in conjunction with the MATLAB implementations.

Caveats using the MATLAB compiler

When using **MOSEK** with the MATLAB compiler it is necessary manually:

- to remove `mosekopt.m` before compilation,
- copy the MEX file to the directory with MATLAB binary files and
- copy the `mosekopt.m` file back after compilation.

10.2 Names

All elements of an optimization problem in **MOSEK** (objective, constraints, variables, etc.) can be given names. Assigning meaningful names to variables and constraints makes it much easier to understand and debug optimization problems dumped to a file. On the other hand, note that assigning names can substantially increase setup time, so it should be avoided in time-critical applications.

Names of various elements of the problem are assigned using the *names* structure within an optimization problem specification *prob*.

10.3 Multithreading

Parallelization

The interior-point and mixed-integer optimizers in **MOSEK** are parallelized. By default **MOSEK** will automatically select the number of threads. However, the maximum number of threads allowed can be changed by setting the parameter *MSK_IPAR_NUM_THREADS* and related parameters. This should never exceed the number of cores. See Sec. 13 and Sec. 13.4 for more details.

The speed-up obtained when using multiple threads is highly problem and hardware dependent. We recommend experimenting with various thread numbers to determine the optimal settings. For small problems using multiple threads may be counter-productive because of the associated overhead.

Determinism

By default the optimizer is run-to-run deterministic, which means that it will return the same answer each time it is run on the same machine with the same input, the same parameter settings (including number of threads) and no time limits.

Setting the number of threads

The number of threads the optimizer uses can be changed with the parameter *MSK_IPAR_NUM_THREADS*. If one process will perform multiple optimizations, this value should be set before the first optimization. After that the thread pool is reserved and fixed in size, so subsequent changes to *MSK_IPAR_NUM_THREADS* will have no effect. The only possibility at that point is to switch between multi-threaded and single-threaded interior-point optimization using the parameter *MSK_IPAR_INTPNT_MULTI_THREAD*.

The MATLAB Parallel Computing Toolbox

Running **MOSEK** with the MATLAB Parallel Computing Toolbox requires multiple **MOSEK** licenses, since each thread runs a separate instance of the **MOSEK** optimizer. Each thread thus requires a **MOSEK** license.

10.4 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 any **MOSEK** function involving optimization, as for instance *mosekopt*, is called the first time and
- it is returned when MATLAB is terminated.

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 *MSK_IPAR_CACHE_LICENSE* to *"MSK_OFF"* will force **MOSEK** to return the license token immediately after the optimization completed.
- Setting the parameter *MSK_IPAR_LICENSE_WAIT* will force **MOSEK** to wait until a license token becomes available instead of returning with an error.
- All licenses currently checked out and not in use can be released on demand using the `nokeepenv` command of *mosekopt*.

```
mosekopt('nokeepenv');
```

Chapter 11

Case Studies

In this section we present some case studies in which the Optimization Toolbox for MATLAB is used to solve real-life applications. These examples involve some more advanced modeling skills and possibly some input data. The user is strongly recommended to first read the basic tutorials of Sec. 6 before going through these advanced case studies.

- *Portfolio Optimization*
  - **Keywords:** Markowitz model, variance, risk, efficient frontier, transaction cost, market impact cost, cardinality constraints
  - **Type:** Conic Quadratic, Power Cone, Mixed-Integer
- *Least squares and other norm minimization problems*
  - **Keywords:** Least squares, regression, 2-norm, 1-norm, p-norm, ridge, lasso
  - **Type:** Conic Quadratic, Power Cone
- *Robust linear optimization*
  - **Keywords:** Robust optimization, ellipsoidal uncertainty
  - **Type:** Conic Quadratic

11.1 Portfolio Optimization

In this section the Markowitz portfolio optimization problem and variants are implemented using Optimization Toolbox for MATLAB.

- *Basic Markowitz model*
- *Efficient frontier*
- *Factor model and efficiency*
- *Market impact costs*
- *Transaction costs*
- *Cardinality constraints*

11.1.1 The Basic Model

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

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

and covariance

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

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

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

and variance

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

The standard deviation

$$\sqrt{x^T \Sigma x}$$

is usually associated with risk.

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

$$\begin{array}{lrcl} \text{maximize} & \mu^T x & & \\ \text{subject to} & e^T x & = & w + e^T x^0, \\ & x^T \Sigma x & \leq & \gamma^2, \\ & x & \geq & 0. \end{array} \tag{11.1}$$

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

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

Since e is the vector of all ones then

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

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

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

This leads to the first constraint

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

The second constraint

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

ensures that the variance, is bounded by the parameter γ^2. Therefore, γ specifies an upper bound of the standard deviation (risk) the investor is willing to undertake. Finally, the constraint

$$x_j \geq 0$$

excludes the possibility of short-selling. This constraint can of course be excluded if short-selling is allowed.

The covariance matrix Σ is positive semidefinite by definition and therefore there exist a matrix G such that

$$\Sigma = GG^T. \tag{11.2}$$

In general the choice of G is **not** unique and one possible choice of G is the Cholesky factorization of Σ. However, in many cases another choice is better for efficiency reasons as discussed in Sec. 11.1.3. For a given G we have that

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

Hence, we may write the risk constraint as

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

or equivalently

$$(\gamma, G^T x) \in \mathcal{Q}^{n+1},$$

where $\mathcal{Q}^{n+1}$ is the $(n+1)$-dimensional quadratic cone. Therefore, problem (11.1) can be written as

$$\begin{array}{lrcl} \text{maximize} & \mu^T x & & \\ \text{subject to} & e^T x & = & w + e^T x^0, \\ & (\gamma, G^T x) & \in & \mathcal{Q}^{n+1}, \\ & x & \geq & 0, \end{array} \tag{11.3}$$

which is a conic quadratic optimization problem that can easily be formulated and solved with Optimization Toolbox for MATLAB. Subsequently we will use the example data

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

and

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

This implies

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

Why a Conic Formulation?

Problem (11.1) is a convex quadratically constrained optimization problem that can be solved directly using **MOSEK**. Why then reformulate it as a conic quadratic optimization problem (11.3)? The main reason for choosing a conic model is that it is more robust and usually solves faster and more reliably. For instance it is not always easy to numerically validate that the matrix Σ in (11.1) is positive semidefinite due to the presence of rounding errors. It is also very easy to make a mistake so Σ becomes indefinite. These problems are completely eliminated in the conic formulation.

Moreover, observe the constraint

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

more numerically robust than

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

for very small and very large values of γ. Indeed, if say $\gamma \approx 10^4$ then $\gamma^2 \approx 10^8$, which introduces a scaling issue in the model. Hence, using conic formulation we work with the standard deviation instead of variance, which usually gives rise to a better scaled model.

Example code

Listing 11.1 demonstrates how the basic Markowitz model (11.3) is implemented.

Listing 11.1: Code implementing problem (11.3).

```
function er = BasicMarkowitz(n,mu,GT,x0,w,gamma)

[rcode, res] = mosekopt('symbcon');
prob = [];

% Objective vector - expected return
prob.c = mu;

% The budget constraint  - e'x = w + sum(x0)
prob.a = ones(1,n);
prob.blc = w + sum(x0);
prob.buc = w + sum(x0);

% Bounds exclude shortselling
prob.blx = zeros(n,1);
prob.bux = inf*ones(n,1);

% An affine conic constraint: [gamma, GT*x] in quadratic cone
prob.f = sparse([ zeros(1,n); GT ]);
prob.g = [gamma; zeros(n,1)];
prob.cones = [ res.symbcon.MSK_CT_QUAD n+1 ];

% Maximize problem and return the objective value
[rcode,res] = mosekopt('maximize echo(0)', prob, []);
x = res.sol.itr.xx;
er = mu'*x;
```

The source code should be self-explanatory except perhaps for

```
prob.f = sparse([ zeros(1,n); GT ]);
prob.g = [gamma; zeros(n,1)];
prob.cones = [ res.symbcon.MSK_CT_QUAD n+1 ];
```

where the constraint

$$(\gamma, G^T x) \in \mathcal{Q}^{n+1}$$

is created as an *affine conic constraint format* of the form $Fx + g \in \mathcal{K}$, in this specific case:

$$\begin{bmatrix} 0 \\ G^T \end{bmatrix} x + \begin{bmatrix} \gamma \\ 0 \end{bmatrix} \in \mathcal{Q}^{n+1}.$$

11.1.2 The Efficient Frontier

The portfolio computed by the Markowitz model is efficient in the sense that there is no other portfolio giving a strictly higher return for the same amount of risk. An efficient portfolio is also sometimes called a Pareto optimal portfolio. Clearly, an investor should only invest in efficient portfolios and therefore it may be relevant to present the investor with all efficient portfolios so the investor can choose the portfolio that has the desired tradeoff between return and risk.

Given a nonnegative α the problem

$$\begin{array}{ll} \text{maximize} & \mu^T x - \alpha x^T \Sigma x \\ \text{subject to} & e^T x = w + e^T x^0, \\ & x \geq 0. \end{array} \tag{11.4}$$

is one standard way to trade the expected return against penalizing variance. Note that, in contrast to the previous example, we explicitly use the variance ($\|G^T x\|_2^2$) rather than standard deviation ($\|G^T x\|_2$), therefore the conic model includes a rotated quadratic cone:

$$
\begin{array}{lrcll}
\text{maximize} & \mu^T x - \alpha s & & & \\
\text{subject to} & e^T x & = & w + e^T x^0, & \\
& (s, 0.5, G^T x) & \in & \mathcal{Q}_r^{n+2} & \text{(equiv. to } s \geq \|G^T x\|_2^2 = x^T \Sigma x), \\
& x & \geq & 0. &
\end{array}
\tag{11.5}
$$

The parameter α specifies the tradeoff between expected return and variance. Ideally the problem (11.4) should be solved for all values $\alpha \geq 0$ but in practice it is impossible. Using the example data from Sec. 11.1.1, the optimal values of return and variance for several values of α are shown in the figure.

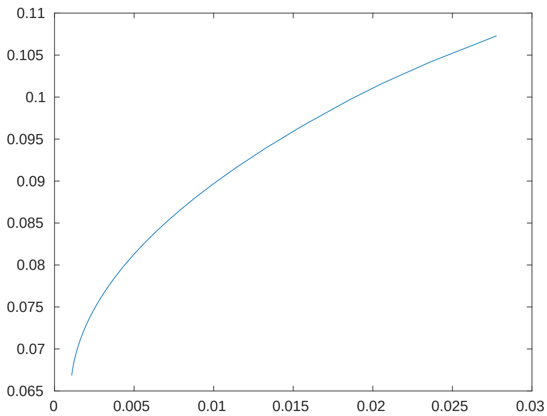

Fig. 11.1: The efficient frontier for the sample data.

Example code

Listing 11.2 demonstrates how to compute the efficient portfolios for several values of α.

Listing 11.2: Code for the computation of the efficient frontier based on problem (11.4).

```
function frontier = EfficientFrontier(n,mu,GT,x0,w,alphas)

frontier = [];
[rcode, res] = mosekopt('symbcon');
prob = [];

% The budget constraint in terms of variables [x; s]
prob.a = [ones(1,n), 0.0];
prob.blc = w + sum(x0);
prob.buc = w + sum(x0);

% No shortselling
prob.blx = [zeros(n,1); -inf];
prob.bux = inf*ones(n+1,1);

% An affine conic constraint: [s, 0.5, GT*x] in rotated quadratic cone
% In matrix form
% [ 0   1]   [ x ]      [ 0   ]
% [ 0   0]   [   ]   +  [ 0.5 ]   \in Q_r
% [ GT  0]   [ s ]      [ 0   ]
prob.f = sparse([ [zeros(1,n), 1.0]; zeros(1, n+1); [GT, zeros(n,1)] ]);
prob.g = [ 0; 0.5; zeros(n, 1) ]
prob.cones = [ res.symbcon.MSK_CT_RQUAD n+2 ];

for alpha = alphas
    % Objective mu'*x - alpha*s
    prob.c = [mu; -alpha];

    [rcode,res] = mosekopt('maximize echo(0)',prob,[]);
    x = res.sol.itr.xx(1:n);
    s = res.sol.itr.xx(n+1);

    frontier = [frontier; [alpha, mu'*x, s] ];
end
```

11.1.3 Factor model and efficiency

In practice it is often important to solve the portfolio problem very quickly. Therefore, in this section we discuss how to improve computational efficiency at the modeling stage.

The computational cost is of course to some extent dependent on the number of constraints and variables in the optimization problem. However, in practice a more important factor is the sparsity: the number of nonzeros used to represent the problem. Indeed it is often better to focus on the number of nonzeros in G see (11.2) and try to reduce that number by for instance changing the choice of G.

In other words if the computational efficiency should be improved then it is always good idea to start with focusing at the covariance matrix. As an example assume that

$$\Sigma = D + VV^T$$

where D is a positive definite diagonal matrix. Moreover, V is a matrix with n rows and p columns. Such a model for the covariance matrix is called a factor model and usually p is much smaller than n. In practice p tends to be a small number independent of n, say less than 100.

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

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

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 storage requirements are reduced by a factor of n.

The example above exploits the so-called factor structure and demonstrates that an alternative choice of G may lead to a significant reduction in the amount of storage used to represent the problem. This will in most cases also lead to a significant reduction in the solution time.

The lesson to be learned is that it is important to investigate how the covariance matrix is formed. Given this knowledge it might be possible to make a special choice for G that helps reducing the storage requirements and enhance the computational efficiency. More details about this process can be found in *[And13]*.

11.1.4 Slippage Cost

The basic Markowitz model assumes that there are no costs associated with trading the assets and that the returns of the assets are independent of the amount traded. Neither of those assumptions is usually valid in practice. Therefore, a more realistic model is

$$\begin{array}{lrcl} \mbox{maximize} & \mu^T x & & \\ \mbox{subject to} & e^T x + \sum_{j=1}^n T_j(\Delta x_j) & = & w + e^T x^0, \\ & x^T \Sigma x & \leq & \gamma^2, \\ & x & \geq & 0. \end{array} \tag{11.6}$$

Here Δx_j is the change in the holding of asset j i.e.

$$\Delta x_j = x_j - x_j^0$$

and $T_j(\Delta x_j)$ specifies the transaction costs when the holding of asset j is changed from its initial value. In the next two sections we show two different variants of this problem with two nonlinear cost functions T.

11.1.5 Market Impact Costs

If the initial wealth is fairly small and no short selling is allowed, then the holdings will be small and the traded amount of each asset must also be small. Therefore, it is reasonable to assume that the prices of the assets are independent of the amount traded. However, if a large volume of an asset is sold or purchased, the price, and hence return, can be expected to change. This effect is called market impact costs. It is common to assume that the market impact cost for asset j can be modeled by

$$T_j(\Delta x_j) = m_j |\Delta x_j|^{3/2}$$

where m_j is a constant that is estimated in some way by the trader. See *[GK00]* [p. 452] for details. From the Modeling Cookbook we know that $t \geq |z|^{3/2}$ can be modeled directly using the power cone $\mathcal{P}_3^{2/3,1/3}$:

$$\{(t, z) : t \geq |z|^{3/2}\} = \{(t, z) : (t, 1, z) \in \mathcal{P}_3^{2/3,1/3}\}$$

Hence, it follows that $\sum_{j=1}^n T_j(\Delta x_j) = \sum_{j=1}^n m_j |x_j - x_j^0|^{3/2}$ can be modeled by $\sum_{j=1}^n m_j t_j$ under the constraints

$$\begin{array}{rl} z_j & = |x_j - x_j^0|, \\ (t_j, 1, z_j) & \in \mathcal{P}_3^{2/3,1/3}. \end{array}$$

Unfortunately this set of constraints is nonconvex due to the constraint

$$z_j = |x_j - x_j^0| \tag{11.7}$$

but in many cases the constraint may be replaced by the relaxed constraint

$$z_j \geq |x_j - x_j^0|, \tag{11.8}$$

For instance if the universe of assets contains a risk free asset then

$$z_j > |x_j - x_j^0| \tag{11.9}$$

cannot hold for an optimal solution.

If the optimal solution has the property (11.9) then the market impact cost within the model is larger than the true market impact cost and hence money are essentially considered garbage and removed by generating transaction costs. This may happen if a portfolio with very small risk is requested because the only way to obtain a small risk is to get rid of some of the assets by generating transaction costs. We generally assume that this is not the case and hence the models (11.7) and (11.8) are equivalent.

The above observations lead to

$$\begin{array}{lrcll} \text{maximize} & \mu^T x & & & \\ \text{subject to} & e^T x + m^T t & = & w + e^T x^0, & \\ & (\gamma, G^T x) & \in & \mathcal{Q}^{n+1}, & \\ & (t_j, 1, x_j - x_j^0) & \in & \mathcal{P}_3^{2/3,1/3}, & j = 1, \ldots, n, \\ & x & \geq & 0. & \end{array} \tag{11.10}$$

The revised budget constraint

$$e^T x + m^T t = w + e^T x^0$$

specifies that the initial wealth covers the investment and the transaction costs. It should be mentioned that transaction costs of the form

$$t_j \geq |z_j|^p$$

where $p > 1$ is a real number can be modeled with the power cone as

$$(t_j, 1, z_j) \in \mathcal{P}_3^{1/p, 1-1/p}.$$

See the Modeling Cookbook for details.

Example code

Listing 11.3 demonstrates how to compute an optimal portfolio when market impact cost are included.

Listing 11.3: Implementation of model (11.10).

```
function [x, t] = MarkowitzWithMarketImpact(n,mu,GT,x0,w,gamma,m)

[rcode, res] = mosekopt('symbcon');

% unrolled variable ordered as (x, t)
prob = [];
prob.c = [mu; zeros(n,1)];

In = speye(n);
On = sparse([],[],[],n,n);

% Linear part
% [ e' m' ]  * [ x; t ]  =   w + e'*x0
prob.a    = [ ones(1,n), m' ];
prob.blc = [ w + sum(x0) ];
prob.buc = [ w + sum(x0) ];

% No shortselling and no other bounds
prob.blx = [ zeros(n,1); -inf*ones(n,1) ];
prob.bux = inf*ones(2*n,1);

% Affine conic constraints representing [ gamma, GT*x ] in quadratic cone
```

(continues on next page)

(continued from previous page)

```
prob.f = sparse([ zeros(1,2*n); [GT On] ]);
prob.g = [gamma; zeros(n,1)];
prob.cones = [ res.symbcon.MSK_CT_QUAD n+1 ];

% Extend the affine conic constraints
% with power cones representing t(i) >= |x(i)-x0(i)|^1.5
fi = [];
fj = [];
g  = [];
fv = repmat([1; 1], n, 1);
for k=1:n
    fi = [fi; 3*k-2; 3*k];
    fj = [fj; n+k; k];
    g  = [g; 0; 1; -x0(k)];
end
prob.f = [prob.f; sparse(fi, fj, fv)];
prob.g = [prob.g; g];
prob.cones = [prob.cones repmat([res.symbcon.MSK_CT_PPOW, 3, 2, 2.0, 1.0], 1, n) ];

[rcode,res] = mosekopt('maximize echo(0)',prob,[]);

x = res.sol.itr.xx(1:n);
t = res.sol.itr.xx(n+(1:n));
```

In the last part of the code we extend the affine conic constraint with triples of the form $(t_k, 1, x_k - x_k^0)$. Such a triple is constructed as an affine conic constraint with:

$$\begin{bmatrix} e_{n+k}^T \\ 0 \\ e_k^T \end{bmatrix} \cdot \begin{bmatrix} x \\ t \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ -x_k^0 \end{bmatrix}$$

where e_j denotes the vector of length $2n$ with 1 at position j and 0 otherwise. Membership of a sequence of triples in power cones $\mathcal{P}_3^{2/3,1/3}$ is specified with the syntax:

```
prob.cones = [prob.cones repmat([res.symbcon.MSK_CT_PPOW, 3, 2, 2.0, 1.0], 1, n) ];
```

Note that the construction `[res.symbcon.MSK_CT_PPOW, d, 2, a, b]` creates a power done of dimension d with exponents

$$\frac{a}{a+b}, \frac{b}{a+b}.$$

11.1.6 Transaction Costs

Now assume there is a cost associated with trading asset j given by

$$T_j(\Delta x_j) = \begin{cases} 0, & \Delta x_j = 0, \\ f_j + g_j|\Delta x_j|, & \text{otherwise.} \end{cases}$$

Hence, whenever asset j is traded we pay a fixed setup cost f_j and a variable cost of g_j per unit traded. Given the assumptions about transaction costs in this section problem (11.6) may be formulated as

$$\begin{array}{lrcll} \text{maximize} & \mu^T x & & & \\ \text{subject to} & e^T x + f^T y + g^T z & = & w + e^T x^0, & \\ & (\gamma, G^T x) & \in & \mathcal{Q}^{n+1}, & \\ & z_j & \geq & x_j - x_j^0, & j = 1, \ldots, n, \\ & z_j & \geq & x_j^0 - x_j, & j = 1, \ldots, n, \\ & z_j & \leq & U_j y_j, & j = 1, \ldots, n, \\ & y_j & \in & \{0, 1\}, & j = 1, \ldots, n, \\ & x & \geq & 0. & \end{array} \tag{11.11}$$

First observe that

$$z_j \geq |x_j - x_j^0| = |\Delta x_j|.$$

We choose U_j as some a priori upper bound on the amount of trading in asset j and therefore if $z_j > 0$ then $y_j = 1$ has to be the case. This implies that the transaction cost for asset j is given by

$$f_j y_j + g_j z_j.$$

Example code

The following example code demonstrates how to compute an optimal portfolio when transaction costs are included.

Listing 11.4: Code solving problem (11.11).

```
function [x, z, y] = MarkowitzWithTransactionsCost(n,mu,GT,x0,w,gamma,f,g)

[rcode, res] = mosekopt('symbcon');

% Upper bound on the traded amount
u = w+sum(x0);

% unrolled variable ordered as (x, z, y)
prob = [];
prob.c = [mu; zeros(2*n,1)];
In = speye(n);
On = sparse([],[],[],n,n);

% Linear constraints
% [ e'  g'  f' ]    [ x ]  =   w + e'*x0
% [ I  -I   0  ] * [ z ]  <=  x0
% [ I   I   0  ]    [ y ]  >=  x0
% [ 0   I  -U  ]           <=  0
prob.a   = [ [ones(1,n), g', f']; In -In On; In In On; On In -u*In ];
prob.blc = [ w + sum(x0); -inf*ones(n,1); x0; -inf*ones(n,1) ];
prob.buc = [ w + sum(x0); x0; inf*ones(n,1); zeros(n,1) ];

% No shortselling and the linear bound 0 <= y <= 1
prob.blx = [ zeros(n,1); -inf*ones(n,1); zeros(n,1) ];
prob.bux = [ inf*ones(2*n,1); ones(n,1) ];

% Affine conic constraints representing [ gamma, GT*x ] in quadratic cone
prob.f = sparse([ zeros(1,3*n); [GT On On];  ]);
prob.g = [gamma; zeros(n,1)];
prob.cones = [ res.symbcon.MSK_CT_QUAD n+1 ];

% Demand y to be integer (hence binary)
prob.ints.sub = 2*n+(1:n);

[rcode,res] = mosekopt('maximize echo(0)',prob,[]);

x = res.sol.int.xx(1:n);
z = res.sol.int.xx(n+(1:n));
y = res.sol.int.xx(2*n+(1:n));
```

11.1.7 Cardinality constraints

Another method to reduce costs involved with processing transactions is to only change positions in a small number of assets. In other words, at most k of the differences $|\Delta x_j| = |x_j - x_j^0|$ are allowed to be non-zero, where k is (much) smaller than the total number of assets n.

This type of constraint can be again modeled by introducing a binary variable y_j which indicates if $\Delta x_j \neq 0$ and bounding the sum of y_j. The basic Markowitz model then gets updated as follows:

$$
\begin{array}{lrcll}
\text{maximize} & \mu^T x & & & \\
\text{subject to} & e^T x & = & w + e^T x^0, & \\
 & (\gamma, G^T x) & \in & \mathcal{Q}^{n+1}, & \\
 & z_j & \geq & x_j - x_j^0, & j = 1, \ldots, n, \\
 & z_j & \geq & x_j^0 - x_j, & j = 1, \ldots, n, \\
 & z_j & \leq & U_j y_j, & j = 1, \ldots, n, \\
 & y_j & \in & \{0, 1\}, & j = 1, \ldots, n, \\
 & e^T y & \leq & k, & \\
 & x & \geq & 0, &
\end{array}
\tag{11.12}
$$

were U_j is some a priori chosen upper bound on the amount of trading in asset j.

Example code

The following example code demonstrates how to compute an optimal portfolio with cardinality bounds.

Listing 11.5: Code solving problem (11.12).

```
function x = MarkowitzWithCardinality(n,mu,GT,x0,w,gamma,k)

[rcode, res] = mosekopt('symbcon');

% Upper bound on the traded amount
u = w+sum(x0);

% unrolled variable ordered as (x, z, y)
prob = [];
prob.c = [mu; zeros(2*n,1)];
In = speye(n);
On = sparse([],[],[],n,n);

% Linear constraints
% [ e'  0    0  ]          =   w + e'*x0
% [ I  -I    0  ]   [ x ]  <=  x0
% [ I   I    0  ] * [ z ]  >=  x0
% [ 0   I   -U  ]   [ y ]  <=  0
% [ 0   0    e' ]          <=  k
prob.a   = [ [ones(1,n), zeros(1,2*n)]; In -In On; In In On; On In -u*In; zeros(1,2*n) ones(1,
↪n) ];
prob.blc = [ w + sum(x0); -inf*ones(n,1); x0; -inf*ones(n,1); 0 ];
prob.buc = [ w + sum(x0); x0; inf*ones(n,1); zeros(n,1); k ];

% No shortselling and the linear bound 0 <= y <= 1
prob.blx = [ zeros(n,1); -inf*ones(n,1); zeros(n,1) ];
prob.bux = [ inf*ones(2*n,1); ones(n,1) ];

% Affine conic constraints representing [ gamma, GT*x ] in quadratic cone
prob.f = sparse([ zeros(1,3*n); [GT On On];  ]);
prob.g = [gamma; zeros(n,1)];
prob.cones = [ res.symbcon.MSK_CT_QUAD n+1 ];

% Demand y to be integer (hence binary)
prob.ints.sub = 2*n+(1:n);

[rcode,res] = mosekopt('maximize echo(0)',prob,[]);

x = res.sol.int.xx(1:n);
```

If we solve our running example with $k = 1, 2, 3$ then we get the following solutions, with increasing expected returns:

```
Bound:  1  Expected return: 0,0627   Solution: 0,0000 0,0000 1,0000
Bound:  2  Expected return: 0,0669   Solution: 0,0939 0,0000 0,9061
Bound:  3  Expected return: 0,0685   Solution: 0,1010 0,1156 0,7834
```

11.2 Least Squares and Other Norm Minimization Problems

A frequently occurring problem in statistics and in many other areas of science is a norm minimization problem

$$\begin{array}{ll} \text{minimize} & \|Fx - g\|, \\ \text{subject to} & Ax = b, \end{array} \tag{11.13}$$

where $x \in \mathbb{R}^n$ and of course we can allow other types of constraints. The objective can involve various norms: infinity norm, 1-norm, 2-norm, p-norms and so on. For instance the most popular case of the 2-norm corresponds to the least squares linear regression, since it is equivalent to minimization of $\|Fx - g\|_2^2$.

11.2.1 Least squares, 2-norm

In the case of the 2-norm we specify the problem directly in conic quadratic form

$$\begin{array}{ll} \text{minimize} & t, \\ \text{subject to} & (t, Fx - g) \in \mathcal{Q}^{k+1}, \\ & Ax = b. \end{array} \tag{11.14}$$

The first constraint of the problem can be represented as an affine conic constraint. This leads to the following model.

Listing 11.6: Script solving problem (11.14)

```
% Least squares regression
% minimize \|Fx-g\|_2
function x = norm_lse(F,g,A,b)
clear prob;
[r, res] = mosekopt('symbcon');
n = size(F,2);
k = size(g,1);
m = size(A,1);

% Linear constraints in [x; t]
prob.a   = [A, zeros(m,1)];
prob.buc = b;
prob.blc = b;
prob.blx = -inf*ones(n+1,1);
prob.bux = inf*ones(n+1,1);
prob.c   = [zeros(n,1); 1];

% Affine conic constraint
prob.f = sparse([zeros(1,n), 1; F, zeros(k,1)]);
prob.g = [0; -g];
prob.cones = [ res.symbcon.MSK_CT_QUAD k+1 ];

% Solve
[r, res] = mosekopt('minimize echo(0)', prob);
x = res.sol.itr.xx(1:n);
end
```

11.2.2 Ridge regularisation

Regularisation is classically applied to reduce the impact of outliers and to control overfitting. In the conic version of *ridge (Tychonov) regression* we consider the problem

$$\begin{array}{ll} \text{minimize} & \|Fx-g\|_2 + \gamma\|x\|_2, \\ \text{subject to} & Ax = b, \end{array} \tag{11.15}$$

which can be written explicitly as

$$\begin{array}{ll} \text{minimize} & t_1 + \gamma t_2, \\ \text{subject to} & (t_1, Fx-g) \in \mathcal{Q}^{k+1}, \\ & (t_2, x) \in \mathcal{Q}^{n+1}, \\ & Ax = b. \end{array} \tag{11.16}$$

The implementation is a small extension of that from the previous section.

Listing 11.7: Script solving problem (11.16)

```
% Least squares regression with regularization
% minimize \|Fx-g\|_2 + gamma*\|x\|_2
function x = norm_lse_reg(F,g,A,b,gamma)
clear prob;
[r, res] = mosekopt('symbcon');
n = size(F,2);
k = size(g,1);
m = size(A,1);

% Linear constraints in [x; t1; t2]
prob.a   = [A, zeros(m,2)];
prob.buc = b;
prob.blc = b;
prob.blx = -inf*ones(n+2,1);
prob.bux = inf*ones(n+2,1);
prob.c   = [zeros(n,1); 1; gamma];

% Affine conic constraint
prob.f = sparse([zeros(1,n),        1, 0; ...
                 F,            zeros(k,2); ...
                 zeros(1,n),        0, 1; ...
                 eye(n),       zeros(n,2) ]);
prob.g = [0; -g; zeros(n+1,1)];
prob.cones = [ res.symbcon.MSK_CT_QUAD k+1 res.symbcon.MSK_CT_QUAD n+1 ];

% Solve
[r, res] = mosekopt('minimize echo(0)', prob);
x = res.sol.itr.xx(1:n);
end
```

Note that classically least squares problems are formulated as quadratic problems and then the objective function would be written as

$$\|Fx-g\|_2^2 + \gamma\|x\|_2^2.$$

This version can easily be obtained by replacing the quadratic cone with an appropriate rotated quadratic cone in (11.16). Then they core of the implementation would change as follows:

Listing 11.8: Script solving classical quadratic ridge regression

```
prob.f = sparse([zeros(1,n),        1, 0; ...
                 zeros(1,n+2)           ; ...
                 F,            zeros(k,2); ...
                 zeros(1,n),        0, 1; ...
```

(continues on next page)

(continued from previous page)

```
                  zeros(1,n+2)           ; ...
                  eye(n),      zeros(n,2) ]);
prob.g = [0; 0.5; -g; 0; 0.5; zeros(n,1)];
prob.cones = [ res.symbcon.MSK_CT_RQUAD k+2 res.symbcon.MSK_CT_RQUAD n+2 ];
```

Fig. 11.2 shows the solution to a polynomial fitting problem for a few variants of least squares regression with and without ridge regularization.

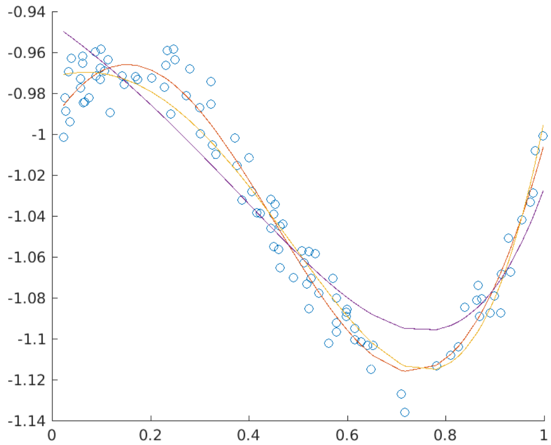

Fig. 11.2: Three fits to a dataset at various levels of regularization.

11.2.3 Lasso regularization

In *lasso* or *least absolute shrinkage and selection operator* the regularization term is the 1-norm of the solution

$$\begin{array}{ll} \text{minimize} & \|Fx-g\|_2 + \gamma\|x\|_1, \\ \text{subject to} & Ax=b. \end{array} \tag{11.17}$$

This variant typically tends to give preference to sparser solutions, i.e. solutions where only a few elements of x are nonzero, and therefore it is used as an efficient approximation to the cardinality constrained problem with an upper bound on the 0-norm of x. To see how it works we first implement (11.17) adding the constraint $t \geq \|x\|_1$ as a series of linear constraints

$$u_i \geq -x_i, \; u_i \geq x_i, \; t \geq \sum u_i,$$

so that eventually the problem becomes

$$\begin{array}{ll} \text{minimize} & t_1 + \gamma t_2, \\ \text{subject to} & u + x \geq 0, \\ & u - x \geq 0, \\ & t_2 - e^T u \geq 0, \\ & Ax = b, \\ & (t_1, Fx-g) \in \mathcal{Q}^{k+1}. \end{array}$$

Listing 11.9: Script solving problem (11.17)

```
% Least squares regression with lasso regularization
% minimize \|Fx-g\|_2 + gamma*\|x\|_1
function x = norm_lse_lasso(F,g,A,b,gamma)
clear prob;
```

(continues on next page)

(continued from previous page)

```
[r, res] = mosekopt('symbcon');
n = size(F,2);
k = size(g,1);
m = size(A,1);

% Linear constraints in [x; u; t1; t2]
prob.a   = [A,          zeros(m,n+2)       ; ...
            eye(n),     eye(n), zeros(n,2); ...
            -eye(n),    eye(n), zeros(n,2); ...
            zeros(1,n) -ones(1,n), 0, 1 ];
prob.buc = [b; inf*ones(2*n+1,1)];
prob.blc = [b; zeros(2*n+1,1)];
prob.blx = -inf*ones(2*n+2,1);
prob.bux = inf*ones(2*n+2,1);
prob.c   = [zeros(2*n,1); 1; gamma];

% Affine conic constraint
prob.f = sparse([zeros(1,2*n), 1, 0; F, zeros(k,n+2)]);
prob.g = [0; -g];
prob.cones = [ res.symbcon.MSK_CT_QUAD k+1 ];

% Solve
[r, res] = mosekopt('minimize echo(0)', prob);
x = res.sol.itr.xx(1:n);
end
```

The sparsity pattern of the solution of a large random regression problem can look for example as follows:

```
Lasso regularization
Gamma 0.0100  density 99%   |Fx-g|_2: 54.3722
Gamma 0.1000  density 87%   |Fx-g|_2: 54.3939
Gamma 0.3000  density 67%   |Fx-g|_2: 54.5319
Gamma 0.6000  density 40%   |Fx-g|_2: 54.8379
Gamma 0.9000  density 26%   |Fx-g|_2: 55.0720
Gamma 1.3000  density 12%   |Fx-g|_2: 55.1903
```

11.2.4 p-norm minimization

Now we consider the minimization of the p-norm defined for $p > 1$ as

$$\|y\|_p = \left(\sum_i |y_i|^p\right)^{1/p}. \tag{11.18}$$

We have the optimization problem:

$$\begin{array}{ll} \text{minimize} & \|Fx-g\|_p, \\ \text{subject to} & Ax = b. \end{array} \tag{11.19}$$

Increasing the value of p forces stronger penalization of outliers as ultimately, when $p \to \infty$, the p-norm $\|y\|_p$ converges to the infinity norm $\|y\|_\infty$ of y. According to the Modeling Cookbook the p-norm bound $t \geq \|Fx-g\|_p$ can be added to the model using a sequence of three-dimensional power cones and we obtain an equivalent problem

$$\begin{array}{ll} \text{minimize} & t \\ \text{subject to} & (r_i, t, (Fx-g)_i) \in \mathcal{P}_3^{1/p,1-1/p}, \\ & e^T r = t, \\ & Ax = b. \end{array} \tag{11.20}$$

The power cones can be added one by one to the structure representing affine conic constraints. Each power cone will require one r_i, one copy of t and one row from F and g. An alternative solution is to create the vector

$$[r_1; \ldots; r_k; t; \ldots; t; Fx - g]$$

and then reshuffle its elements into

$$[r_1; t; F_1x - g_1; \ldots; r_k; t; F_kx - g_k]$$

using an appropriate permutation matrix. This approach is demonstrated in the code below.

Listing 11.10: Script solving problem (11.20)

```
% P-norm minimization
% minimize \|Fx-g\|_p
function x = norm_p_norm(F,g,A,b,p)
clear prob;
[r, res] = mosekopt('symbcon');
n = size(F,2);
k = size(g,1);
m = size(A,1);

% Linear constraints in [x; r; t]
prob.a   = [A, zeros(m,k+1); zeros(1,n), ones(1,k), -1];
prob.buc = [b; 0];
prob.blc = [b; 0];
prob.blx = -inf*ones(n+k+1,1);
prob.bux = inf*ones(n+k+1,1);
prob.c   = [zeros(n+k,1); 1];

% Permutation matrix which picks triples (r_i, t, F_ix-g_i)
M = [];
for i=1:3
  M = [M, sparse(i:3:3*k, 1:k, ones(k,1), 3*k, k)];
end

% Affine conic constraint
prob.f = M * sparse([zeros(k,n), eye(k), zeros(k,1); zeros(k,n+k), ones(k,1); F, zeros(k,
↪k+1)]);
prob.g = M * [zeros(2*k,1); -g];
prob.cones = [ repmat([res.symbcon.MSK_CT_PPOW, 3, 2, 1.0, p-1], 1, k) ];

% Solve
[r, res] = mosekopt('minimize echo(0)', prob);
x = res.sol.itr.xx(1:n);
end
```

11.3 Robust linear Optimization

In most linear optimization examples discussed in this manual it is implicitly assumed that the problem data, such as c and A, is known with certainty. However, in practice this is seldom the case, e.g. the data may just be roughly estimated, affected by measurement errors or be affected by random events.

In this section a robust linear optimization methodology is presented which removes the assumption that the problem data is known exactly. Rather it is assumed that the data belongs to some set, i.e. a box or an ellipsoid.

The computations are performed using the **MOSEK** optimization toolbox for MATLAB but could equally well have been implemented using the **MOSEK** API.

This section is co-authored with A. Ben-Tal and A. Nemirovski. For further information about robust linear optimization consult *[BTN00]*, *[BenTalN01]*.

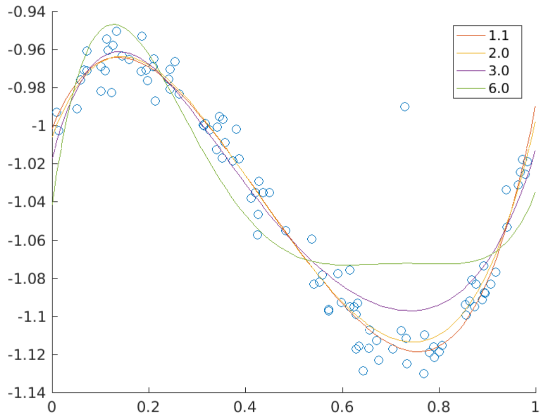

Fig. 11.3: p-norm minimizing fits of a polynomial of degree at most 5 to the data for various values of p.

11.3.1 Introductory Example

Consider the following toy-sized linear optimization problem: A company produces two kinds of drugs, `DrugI` and `DrugII`, containing a specific active agent A, which is extracted from a raw materials that should be purchased on the market. The drug production data are as follows:

Selling price \$ per 1000 packs	6200	6900
Content of agent A gm per 100 packs	0.500	0.600
Production expenses		
\$ per 1000 packs		
Manpower, hours	90.0	100.0
Equipment, hours	40.0	50.0
Operational cost, \$	700	800

There are two kinds of raw materials, RawI and RawII, which can be used as sources of the active agent. The related data is as follows:

Raw material	Purchasing price,	Content of agent A,
RawI	100.00	0.01
RawII	199.90	0.02

Finally, the monthly resources dedicated to producing the drugs are as follows:

Budget,‘	Manpower	Equipment	Capacity of raw materials
100000	2000	800	1000

The problem is to find the production plan which maximizes the profit of the company, i.e. minimize the purchasing and operational costs

$$100 \cdot \texttt{RawI} + 199.90 \cdot \texttt{RawII} + 700 \cdot \texttt{DrugI} + 800 \cdot \texttt{DrugII}$$

and maximize the income

$$6200 \cdot \texttt{DrugI} + 6900 \cdot \texttt{DrugII}$$

The problem can be stated as the following linear programming program:
Minimize

$$-\{100 \cdot \texttt{RawI} + 199.90 \cdot \texttt{RawII} + 700 \cdot \texttt{DrugI} + 800 \cdot \texttt{DrugII}\} + \{6200 \cdot \texttt{DrugI} + 6900 \cdot \texttt{DrugII}\} \quad (11.21)$$

subject to

$$\begin{array}{rcll} 0.01 \cdot \mathtt{RawI} + 0.02 \cdot \mathtt{RawII} - 0.500 \cdot \mathtt{DrugI} - 0.600 \cdot \mathtt{DrugII} & \geq & 0 & (a) \\ \mathtt{RawI} + \mathtt{RawII} & \leq & 1000 & (b) \\ 90.0 \cdot \mathtt{DrugI} + 100.0 \cdot \mathtt{DrugII} & \leq & 2000 & (c) \\ 40.0 \cdot \mathtt{DrugI} + 50.0 \cdot \mathtt{DrugII} & \leq & 800 & (d) \\ 100.0 \cdot \mathtt{RawI} + 199.90 \cdot \mathtt{RawII} + 700 \cdot \mathtt{DrugI} + 800 \cdot \mathtt{DrugII} & \leq & 100000 & (d) \\ \mathtt{RawI}, \mathtt{RawII}, \mathtt{DrugI}, \mathtt{DrugII} & \geq & 0 & (e) \end{array}$$

where the variables are the amounts `RawI`, `RawII` (in kg) of raw materials to be purchased and the amounts `DrugI`, `DrugII` (in 1000 of packs) of drugs to be produced. The objective (11.21) denotes the profit to be maximized, and the inequalities can be interpreted as follows:

- Balance of the active agent.
- Storage restriction.
- Manpower restriction.
- Equipment restriction.
- Ducget restriction.

Listing 11.11 is the MATLAB script which specifies the problem and solves it using the **MOSEK** optimization toolbox:

Listing 11.11: Script *rlo1.m*.

```
function rlo1()

prob.c   = [-100;-199.9;6200-700;6900-800];
prob.a   = sparse([0.01,0.02,-0.500,-0.600;1,1,0,0;
                   0,0,90.0,100.0;0,0,40.0,50.0;100.0,199.9,700,800]);
prob.blc = [0;-inf;-inf;-inf;-inf];
prob.buc = [inf;1000;2000;800;100000];
prob.blx = [0;0;0;0];
prob.bux = [inf;inf;inf;inf];
[r,res]  = mosekopt('maximize',prob);
xx       = res.sol.itr.xx;
RawI     = xx(1);
RawII    = xx(2);
DrugI    = xx(3);
DrugII   = xx(4);

disp(sprintf('*** Optimal value: %8.3f',prob.c'*xx));
disp('*** Optimal solution:');
disp(sprintf('RawI:   %8.3f',RawI));
disp(sprintf('RawII:  %8.3f',RawII));
disp(sprintf('DrugI:  %8.3f',DrugI));
disp(sprintf('DrugII: %8.3f',DrugII));
```

When executing this script, the following is displayed:

Listing 11.12: Output of script *rlo1.m*

```
*** Optimal value: 8819.658
*** Optimal solution:
RawI:     0.000
RawII:  438.789
DrugI:   17.552
DrugII:   0.000
```

We see that the optimal solution promises the company a modest but quite respectful profit of 8.8%. Please note that at the optimal solution the balance constraint is active: the production process utilizes the full amount of the active agent contained in the raw materials.

11.3.2 Data Uncertainty and its Consequences.

Please note that not all problem data can be regarded as *absolutely* reliable; e.g. one can hardly believe that the contents of the active agent in the raw materials are *exactly* the *nominal data* 0.01 gm/kg for `RawI` and 0.02 gm/kg for *RawII*. In reality, these contents definitely vary around the indicated values. A natural assumption here is that the actual contents of the active agent a_i in `RawI` and a_{II} in `RawII` are realizations of random variables somehow distributed around the *nominal contents* $a_i^{\rm n} = 0.01$ and $a_{II}^{\rm n} = 0.02$. To be more specific, assume that a_i drifts in the 0.5% margin of $a_i^{\rm n}$, i.e. it takes with probability 0.5 the values from the interval $a_i^{\rm n}(1 \pm 0.005) = a_i^{\rm n}\{0.00995; 0.01005\}$. Similarly, assume that a_{II} drifts in the 2% margin of $a_{II}^{\rm n}$, taking with probabilities 0.5 the values $a_{II}^{\rm n}(1 \pm 0.02) = a_i^{\rm n}\{0.0196; 0.0204\}$. How do the perturbations of the contents of the active agent affect the production process?

The optimal solution prescribes to purchase 438.8 kg of `RawII` and to produce 17552 packs of DrugI. With the above random fluctuations in the content of the active agent in `RawII`, this production plan, with probability 0.5, will be infeasible – with this probability, the actual content of the active agent in the raw materials will be less than required to produce the planned amount of DrugI. For the sake of simplicity, assume that this difficulty is resolved in the simplest way: when the actual content of the active agent in the raw materials is insufficient, the output of the drug is reduced accordingly. With this policy, the actual production of DrugI becomes a random variable which takes, with probabilities 0.5, the nominal value of 17552 packs and the 2% less value of 17201 packs. These 2% fluctuations in the production affect the profit as well; the latter becomes a random variable taking, with probabilities 0.5, the nominal value 8,820 and the 21% less value 6,929. The expected profit is 7,843, which is by 11% less than the nominal profit 8,820 promised by the optimal solution of the problem.

We see that in our toy example that small (and in reality unavoidable) perturbations of the data may make the optimal solution infeasible, and a straightforward adjustment to the actual solution values may heavily affect the solution quality.

It turns out that the outlined phenomenon is found in many linear programs of practical origin. Usually, in these programs at least part of the data is not known exactly and can vary around its nominal values, and these data perturbations can make the nominal optimal solution – the one corresponding to the nominal data – infeasible. It turns out that the consequences of data uncertainty can be much more severe than in our toy example. The analysis of linear optimization problems from the NETLIB collection[1] reported in *[BTN00]* demonstrates that for 13 of 94 NETLIB problems, already 0.01% perturbations of "clearly uncertain" data can make the nominal optimal solution severely infeasible: with these perturbations, the solution, with a non-negligible probability, violates some of the constraints by 50% and more. It should be added that in the general case, in contrast to the toy example we have considered, there is no evident way to adjust the optimal solution by a small modification to the actual values of the data. Moreover there are cases when such an adjustment is impossible — in order to become feasible for the perturbed data, the nominal optimal solution should be *completely reshaped*.

11.3.3 Robust Linear Optimization Methodology

A natural approach to handling data uncertainty in optimization is offered by the *Robust Optimization Methodology* which, as applied to linear optimization, is as follows.

Uncertain Linear Programs and their Robust Counterparts.

Consider a linear optimization problem

$$\begin{array}{lccccc} \text{minimize} & & & c^T x & & \\ \text{subject to} & l_c & \leq & Ax & \leq & u_c, \\ & l_x & \leq & x & \leq & u_x, \end{array} \tag{11.22}$$

with the data $(c, A, l_c, u_c, l_x, u_x)$, and assume that this data is not known exactly; all we know is that the data varies in a given *uncertainty set* $\mathcal{U}$. The simplest example is the one of *interval uncertainty*, where every data entry can run through a given interval:

$$\begin{array}{rcl} \mathcal{U} & = & \{(c, A, l_c, u_c, l_x, u_x): \\ & & (c^{\rm n} - dc, A^{\rm n} - dA, l_c^{\rm n} - dl_c, u_c^{\rm n} - du_c, l_x^{\rm n} - dl_x, u_x^{\rm n} - du_x) \leq (c, A, l_c, u_c, l_x, u_x) \\ & & \leq (c^{\rm n} + dc, A^{\rm n} + dA, l_c^{\rm n} + dl_c, u_c^{\rm n} + du_c, l_x^{\rm n} + dl_x, u_x^{\rm n} + du_x)\}. \end{array} \tag{11.23}$$

[1] NETLIB is a collection of LP's, mainly of the real world origin, which is a standard benchmark for evaluating LP algorithms

Here

$$(c^{\mathrm{n}}, A^{\mathrm{n}}, l_c^{\mathrm{n}}, u_c^{\mathrm{n}}, l_x^{\mathrm{n}}, u_x^{\mathrm{n}})$$

is the *nominal data*,

$$dc, dA, dl_c, du_c, dl_x, du_x \geq 0$$

is the *data perturbation bounds.* Please note that some of the entries in the data perturbation bounds can be zero, meaning that the corresponding data entries are certain (the expected values equals the actual values).

- The family of instances (11.22) with data running through a given uncertainty set $\mathcal{U}$ is called an *uncertain linear optimization problem.*
- Vector x is called a *robust feasible solution* to an uncertain linear optimization problem, if it remains feasible for all realizations of the data from the uncertainty set, i.e. if

  $$l_c \leq Ax \leq u_c, l_x \leq x \leq u_x$$

  for all

  $$(c, A, l_c, u_c, l_x, u_x) \in \mathcal{U}.$$

- If for some value t we have $c^T x \leq t$ for all realizations of the objective from the uncertainty set, we say that *robust value of the objective* at x does not exceed t.

The Robust Optimization methodology proposes to associate with an uncertain linear program its *robust counterpart* (RC) which is *the problem of minimizing the robust optimal value over the set of all robust feasible solutions*, i.e. the problem

$$\min_{t,x} \left\{ t : c^T x \leq t, l_c \leq Ax \leq u_c, l_x \leq x \leq u_x \forall (c, A, l_c, u_c, l_x, u_x) \in \mathcal{U} \right\}. \tag{11.24}$$

The optimal solution to (11.24) is treated as the *uncertainty-immuned* solution to the original uncertain linear programming program.

Robust Counterpart of an Uncertain Linear Optimization Problem with Interval Uncertainty

In general, the RC (11.24) of an uncertain linear optimization problem is not a linear optimization problem since (11.24) has infinitely many linear constraints. There are, however, cases when (11.24) can be rewritten equivalently as a linear programming program; in particular, this is the case for interval uncertainty (11.23). Specifically, in the case of (11.23), the robust counterpart of uncertain linear program is equivalent to the following linear program in variables x, y, t:

$$\begin{array}{lrclcll}
\text{minimize} & & & t & & & \\
\text{subject to} & & & (c^{\mathrm{n}})^T x + (dc)^T y - t & \leq & 0, & (a) \\
& l_c^{\mathrm{n}} + dl_c & \leq & (A^{\mathrm{n}})x - (dA)y, & & & (b) \\
& & & (A^{\mathrm{n}})x + (dA)y & \leq & u_c^{\mathrm{n}} - du_c, & (c) \\
& 0 & \leq & x + y, & & & (d) \\
& 0 & \leq & -x + y, & & & (e) \\
& l_x^{\mathrm{n}} + dl_x & \leq & x & \leq & u_x^{\mathrm{n}} - du_x, & (f)
\end{array} \tag{11.25}$$

The origin of (11.25) is quite transparent: The constraints $d - e$ in (11.25) linking x and y merely say that $y_i \geq |x_i|$ for all i. With this in mind, it is evident that at every feasible solution to (11.25) the entries in the vector

$$(A^{\mathrm{n}})x - (dA)y$$

are lower bounds on the entries of Ax with A from the uncertainty set (11.23), so that (b) in (11.25) ensures that $l_c \leq Ax$ for all data from the uncertainty set. Similarly, $(c), (a)$ ans f in (11.25) ensure, for all data from the uncertainty set, that $Ax \leq u_c$, $c^T x \leq t$, and that the entries in x satisfy the required lower and upper bounds, respectively.

Please note that at the optimal solution to (11.25), one clearly has $y_j = |x_j|$. It follows that when the bounds on the entries of x impose nonnegativity (nonpositivity) of an entry x_j, then there is no need to introduce the corresponding additional variable y_i — from the very beginning it can be replaced with x_j, if x_j is nonnegative, or with $-x_j$, if x_j is nonpositive.

Another possible formulation of problem (11.25) is the following. Let

$$l_c^{\text{n}} + dl_c = (A^{\text{n}})x - (dA)y - f, f \geq 0$$

then this equation is equivalent to $(a)-(b)$ in (11.25). If $(l_c)_i = -\infty$, then equation i should be dropped from the computations. Similarly,

$$-x + y = g \geq 0$$

is equivalent to (d) in (11.25). This implies that

$$l_c^{\text{n}} + dl_c - (A^{\text{n}})x + f = -(dA)y$$

and that

$$y = g + x$$

Substituting these values into (11.25) gives

$$\begin{array}{lcccc}
\text{minimize} & & t & & \\
\text{subject to} & & (c^{\text{n}})^T x + (dc)^T (g + x) - t & \leq & 0, \\
0 & \leq & f, & & \\
 & & 2(A^{\text{n}})x + (dA)(g + x) + f + l_c^{\text{n}} + dl_c & \leq & u_c^{\text{n}} - du_c, \\
0 & \leq & g, & & \\
0 & \leq & 2x + g, & & \\
l_x^{\text{n}} + dl_x & \leq & x & \leq & u_x^{\text{n}} - du_x,
\end{array}$$

which after some simplifications leads to

$$\begin{array}{lccccc}
\text{minimize} & & t & & & \\
\text{subject to} & & (c^{\text{n}} + dc)^T x + (dc)^T g - t & \leq & 0, & (a) \\
0 & \leq & f, & & & (b) \\
 & & 2(A^{\text{n}} + dA)x + (dA)g + f - (l_c^{\text{n}} + dl_c) & \leq & u_c^{\text{n}} - du_c, & (c) \\
0 & \leq & g, & & & (d) \\
0 & \leq & 2x + g, & & & (e) \\
l_x^{\text{n}} + dl_x & \leq & x & \leq & u_x^{\text{n}} - du_x, & (f)
\end{array}$$

and

$$\begin{array}{lccccc}
\text{minimize} & & t & & & \\
\text{subject to} & & (c^{\text{n}} + dc)^T x + (dc)^T g - t & \leq & 0, & (a) \\
 & & 2(A^{\text{n}} + dA)x + (dA)g + f & \leq & u_c^{\text{n}} - du_c + l_c^{\text{n}} + dl_c, & (b) \\
0 & \leq & 2x + g, & & & (c) \\
0 & \leq & f, & & & (d) \\
0 & \leq & g, & & & (e) \\
l_x^{\text{n}} + dl_x & \leq & x & \leq & u_x^{\text{n}} - du_x. & (f)
\end{array} \tag{11.26}$$

Please note that this problem has more variables but much fewer constraints than (11.25). Therefore, (11.26) is likely to be solved faster than (11.25). Note too that (11.26).b is trivially redundant if $l_x^{\text{n}} + dl_x \geq 0$.

Introductory Example (continued)

Let us apply the Robust Optimization methodology to our drug production example presented in Sec. 11.3.1, assuming that the only uncertain data is the contents of the active agent in the raw materials, and that these contents vary in 0.5% and 2% neighborhoods of the respective nominal values 0.01 and 0.02. With this assumption, the problem becomes an uncertain LP affected by interval uncertainty; the robust counterpart (11.25) of this uncertain LP is the linear program

$$
\begin{array}{lll}
(\text{Drug\_RC}): & & \\
\text{maximize} & & \\
t & & \\
\text{subject to} & & \\
t \leq -100 \cdot \texttt{RawI} - 199.9 \cdot \texttt{RawII} + 5500 \cdot \texttt{DrugI} + 6100 \cdot \texttt{DrugII} & & \\
0.01 \cdot 0.995 \cdot \texttt{RawI} + 0.02 \cdot 0.98 \cdot \texttt{RawII} - 0.500 \cdot \texttt{DrugI} - 0.600 \cdot \texttt{DrugII} & \geq & 0 \\
\texttt{RawI} + \texttt{RawII} & \leq & 1000 \\
90.0 \cdot \texttt{DrugI} + 100.0 \cdot \texttt{DrugII} & \leq & 2000 \\
40.0 \cdot \texttt{DrugI} + 50.0 \cdot \texttt{DrugII} & \leq & 800 \\
100.0 \cdot \texttt{RawI} + 199.90 \cdot \texttt{RawII} + 700 \cdot \texttt{DrugI} + 800 \cdot \texttt{DrugII} & \leq & 100000 \\
\texttt{RawI}, \texttt{RawII}, \texttt{DrugI}, \texttt{DrugII} & \geq & 0
\end{array}
\tag{11.27}
$$

Solving this problem with **MOSEK** we get the following output:

Listing 11.13: Output solving problem (11.27).

```
*** Optimal value: 8294.567
*** Optimal solution:
RawI:    877.732
RawII:     0.000
DrugI:    17.467
DrugII:    0.000
```

We see that the robust optimal solution we have built *costs money* – it promises a profit of just 8, 295 (cf. with the profit of 8, 820 promised by the nominal optimal solution). Please note, however, that the robust optimal solution remains feasible whatever are the realizations of the uncertain data from the uncertainty set in question, while the nominal optimal solution requires adjustment to this data and, with this adjustment, results in the average profit of 7, 843, which is by 5.4% *less* than the profit of ‘ 8,295 *guaranteed* by the robust optimal solution. Note too that the robust optimal solution is significantly different from the nominal one: both solutions prescribe to produce the same drug `DrugI` (in the amounts 17, 467 and 17, 552 packs, respectively) but from different raw materials, `RawI` in the case of the robust solution and `RawII` in the case of the nominal solution. The reason is that although the price per unit of the active agent for `RawII` is sligthly less than for `RawI`, the content of the agent in `RawI` is more stable, so when possible fluctuations of the contents are taken into account, `RawI` turns out to be more profitable than `RawII`.

11.3.4 Random Uncertainty and Ellipsoidal Robust Counterpart

In some cases, it is natural to assume that the perturbations affecting different uncertain data entries are random and independent of each other. In these cases, the robust counterpart based on the interval model of uncertainty seems to be too conservative: Why should we expect that all the data will be simultaneously driven to its most unfavorable values and immune the solution against this highly unlikely situation? A less conservative approach is offered by the *ellipsoidal* model of uncertainty. To motivate this model, let us seseee what happens with a particular linear constraint

$$a^T x \leq b \tag{11.28}$$

at a given candidate solution x in the case when the vector a of coefficients of the constraint is affected by random perturbations:

$$a = a^{\mathrm{n}} + \zeta, \tag{11.29}$$

where a^{n} is the vector of nominal coefficients and ζ is a random perturbation vector with zero mean and covariance matrix V_a. In this case the value of the left-hand side of (11.28), evaluated at a given x, becomes a random variable with the expected value $(a^{\mathrm{n}})^T x$ and the standard deviation $\sqrt{x^T V_a x}$. Now let us act as an engineer who believes that the value of a random variable never exceeds its mean plus 3 times the standard deviation; we do not intend to be that specific and replace *3* in the above rule by a safety parameter Ω which will be in our control. Believing that the value of a random variable *never* exceeds its mean plus Ω times the standard deviation, we conclude that a *safe* version of (11.28) is the inequality

$$(a^{\mathrm{n}})^T x + \Omega\sqrt{x^T V_a x} \leq b. \tag{11.30}$$

The word *safe* above admits a quantitative interpretation: If x satisfies (11.30), one can bound from above the probability of the event that random perturbations (11.29) result in violating the constraint (11.28) evaluated at x. The bound in question depends on what we know about the distribution of ζ, e.g.

- We always have the bound given by the Tschebyshev inequality: x satisfies (11.30) $\Rightarrow$

$$\text{Prob}\left\{a^T x > b\right\} \leq \frac{1}{\Omega^2}.$$

- When ζ is Gaussian, then the Tschebyshev bound can be improved to: x satisfies (11.30) $\Rightarrow$

$$\text{Prob}\left\{a^T x > b\right\} \leq \frac{1}{\sqrt{2\pi}}\int_{\Omega}^{\infty} \exp\{-t^2/2\}dt \leq 0.5\exp\{-\Omega^2/2\}. \tag{11.31}$$

- Assume that $\zeta = D\xi$, where Δ is certain $n \times m$ matrix, and $\xi = (\xi_1, \ldots, \xi_m)^T$ is a random vector with independent coordinates $\xi_1, \ldots, \xi_m$ symmetrically distributed in the segment $[-1, 1]$. Setting $V = DD^T$ (V is a natural *upper bound* on the covariance matrix of ζ), one has: x satisfies (11.30) implies

$$\text{Prob}\left\{a^T x > b\right\} \leq 0.5\exp\{-\Omega^2/2\}. \tag{11.32}$$

Please note that in order to ensure the bounds in (11.31) and (11.32)) to be $\leq 10^{-6}$, it suffices to set $\Omega = 5.13$.

Now, assume that we are given a linear program affected by random perturbations:

$$\begin{array}{lccccl} \text{minimize} & & & [c^{\mathrm{n}} + dc]^T x & & \\ \text{subject to} & (l_c)_i & \leq & [a_i^{\mathrm{n}} + da_i]^T x & \leq & (u_c)_i, i = 1, \ldots, m, \\ & l_x & \leq & x & \leq & u_x, \end{array} \tag{11.33}$$

where $(c^{\mathrm{n}}, \{a_i^{\mathrm{n}}\}_{i=1}^m, l_c, u_c, l_x, u_x)$ are the nominal data, and dc, da_i are random perturbations with zero means[3]. Assume, for the sake of definiteness, that every one of the random perturbations $dc, da_1, \ldots, da_m$ satisfies either the assumption of item 2 or the assumption of item 3, and let $V_c, V_1, \ldots, V_m$ be the corresponding (upper bounds on the) covariance matrices of the perturbations. Choosing a safety parameter Ω and replacing the objective and the bodies of all the constraints by their safe bounds as explained above, we arrive at the following optimization problem:

$$\begin{array}{lccccl} \text{minimize} & & & t & & \\ \text{subject to} & & & [c^{\mathrm{n}}]^T x + \Omega\sqrt{x^T V_c x} & \leq & t, \\ & (l_c)_i & \leq & [a_i^{\mathrm{n}}]^T x - \Omega\sqrt{x^T V_{a_i} x}, & & \\ & & & [a_i^{\mathrm{n}}]^T x + \Omega\sqrt{x^T V_{a_i} x} & \leq & (u_c)_i, i = 1, \ldots, m, \\ & l_x & \leq & x & \leq & u_x. \end{array} \tag{11.34}$$

The relation between problems (11.34) and (11.33) is as follows:

- If (x, t) is a feasible solution of (11.34), then with probability at least

$$p = 1 - (m+1)\exp\{-\Omega^2/2\}$$

x is feasible for randomly perturbed problem (11.33), and t is an upper bound on the objective of (11.33) evaluated at x.

- We see that if Ω is not too small (11.34) can be treated as a "safe version" of (11.33).

[3] For the sake of simplicity, we assume that the bounds l_c, u_c, l_x, u_x are not affected by uncertainty; extensions to the case when it is not so are evident.

On the other hand, it is easily seen that (11.34) is nothing but the robust counterpart of the uncertain linear optimization problem with the nominal data $(c^{\mathrm{n}}, \{a_i^{\mathrm{n}}\}_{i=1}^m, l_c, u_c, l_x, u_x)$ and the row-wise ellipsoidal uncertainty given by the matrices $V_c, V_{a_1}, \ldots, V_{a_m}$. In the corresponding uncertainty set, the uncertainty affects the coefficients of the objective and the constraint matrix only, and the perturbation vectors affecting the objective and the vectors of coefficients of the linear constraints run, independently of each other, through the respective ellipsoids

$$E_c = \left\{ dc = \Omega V_c^{1/2} u : u^T u \le 1 \right\} E_{a_i} = \left\{ da_i = \Omega V_{a_i}^{1/2} u : u^T u \le 1 \right\}, i = 1, \ldots, m.$$

It turns out that in many cases the ellipsoidal model of uncertainty is significantly less conservative and thus better suited for practice, than the interval model of uncertainty.

Last but not least, it should be mentioned that problem (11.34) is equivalent to a conic quadratic program, specifically to the program

$$\begin{array}{lcccl}
\text{minimize} & & t & & \\
\text{subject to} & & [c^{\mathrm{n}}]^T x + \Omega z & \le & t, \\
(l_c)_i & \le & [a_i^{\mathrm{n}}]^T x - \Omega z_i, & & \\
 & & [a_i^{\mathrm{n}}]^T x + \Omega z_i & \le & (u_c)_i, i = 1, \ldots, m, \\
0 & = & w - D_c x & & \\
0 & = & w^i - D_{a_i} x, & & i = 1, \ldots, m, \\
0 & \le & z - \sqrt{w^T w}, & & \\
0 & \le & z_i - \sqrt{(w^i)^T w^i}, & & i = 1, \ldots, m, \\
l_x & \le & x & \le & u_x.
\end{array}$$

where D_c and D_{a_i} are matrices satisfying the relations

$$V_c = D_c^T D_c, V_{a_i} = D_{a_i}^T D_{a_i}, i = 1, \ldots, m.$$

Example: Interval and Ellipsoidal Robust Counterparts of Uncertain Linear Constraint with Independent Random Perturbations of Coefficients

Consider a linear constraint

$$l \le \sum_{j=1}^{n} a_j x_j \le u \tag{11.35}$$

and assume that the a_j coefficients of the body of the constraint are uncertain and vary in intervals $a_j^{\mathrm{n}} \pm \sigma_j$. The worst-case_oriented model of uncertainty here is the interval one, and the corresponding robust counterpart of the constraint is given by the system of linear inequalities

$$\begin{array}{lcccl}
l & \le & \sum_{j=1}^n a_j^{\mathrm{n}} x_j - \sum_{j=1}^n \sigma_j y_j, & & \\
 & & \sum_{j=1}^n a_j^{\mathrm{n}} x_j + \sum_{j=1}^n \sigma_j y_j & \le & u, \\
0 & \le & x_j + y_j, & & \\
0 & \le & -x_j + y_j, & & j = 1, \ldots, n.
\end{array} \tag{11.36}$$

Now, assume that we have reasons to believe that the true values of the coefficients a_j are obtained from their nominal values a_j^{n} by random perturbations, independent for different j and symmetrically distributed in the segments *[-sigma_j,sigma_j]*. With this assumption, we are in the situation of item 3 and can replace the uncertain constraint (11.35) with its ellipsoidal robust counterpart

$$\begin{array}{lcccl}
l & \le & \sum_{j=1}^n a_j^{\mathrm{n}} x_j - \Omega z, & & \\
 & & \sum_{j=1}^n a_j^{\mathrm{n}} x_j + \Omega z & \le & u, \\
0 & \le & z - \sqrt{\sum_{j=1}^n \sigma_j^2 x_j^2}. & &
\end{array} \tag{11.37}$$

Please note that with the model of random perturbations, a vector x satisfying (11.37) satisfies a realization of (11.35) with probability at least $1 - \exp\{\Omega^2/2\}$; for $\Omega = 6$. This probability is $\ge 1 - 1.5 \cdot 10^{-8}$, which for all practical purposes is the same as sayiong that x satisfies all realizations of (11.35). On the other hand, the uncertainty set associated with (11.36) is the box

$$B = \left\{ a = (a_1, \dots, a_n)^T : a_j^{\mathrm{n}} - \sigma_j \le a_j \le a_j^{\mathrm{n}} + \sigma_j, j = 1, \dots, n \right\},$$

while the uncertainty set associated with (11.37) is the ellipsoid

$$E(\Omega) = \left\{ a = (a_1, \dots, a_n)^T : \sum_{j=1}^{n} (a_j - a_j^{\mathrm{n}})^{\frac{2}{\sigma_j^2}} \le \Omega^2 \right\}.$$

For a moderate value of Ω, say $\Omega = 6$, and $n \ge 40$, the ellipsoid $E(\Omega)$ in its diameter, typical linear sizes, volume, etc. is incomparably less than the box B, the difference becoming more dramatic the larger the dimension n of the box and the ellipsoid. It follows that the ellipsoidal robust counterpart (11.37) of the randomly perturbed uncertain constraint (11.35) is much less conservative than the interval robust counterpart (11.36), while ensuring basically the same "robustness guarantees". To illustrate this important point, consider the following numerical examples:

There are n different assets on the market. The return on 1 invested in asset j is a random variable distributed symmetrically in the segment $[\delta_j - \sigma_j, \delta_j + \sigma_j]$, and the returns on different assets are independent of each other. The problem is to distribute ' 1 among the assets in order to get the largest possible total return on the resulting portfolio.

A natural model of the problem is an uncertain linear optimization problem

$$\begin{array}{lccll} \text{maximize} & & \sum_{j=1}^{n} a_j x_j & & \\ \text{subject to} & & \sum_{j=1}^{n} x_j & = & 1, \\ & 0 \le & x_j, & & j = 1, \dots, n. \end{array}$$

where a_j are the uncertain returns of the assets. Both the nominal optimal solution (set all returns a_j equal to their nominal values δ_j) and the risk-neutral Stochastic Programming approach (maximize the expected total return) result in the same solution: Our money should be invested in the most promising asset(s) – the one(s) with the maximal nominal return. This solution, however, can be very unreliable if, as is typically the case in reality, the most promising asset has the largest volatility σ and is in this sense the most risky. To reduce the risk, one can use the Robust Counterpart approach which results in the following optimization problems.

The Interval Model of Uncertainty:

$$\begin{array}{lccll} \text{maximize} & & t & & \\ \text{subject to} & 0 \le & -t + \sum_{j=1}^{n} (\delta_j - \sigma_j) x_j, & & \\ & & \sum_{j=1}^{n} x_j & = & 1, \\ & 0 \le & x_j, & & j = 1, \dots, n \end{array} \tag{11.38}$$

and

The ellipsoidal Model of Uncertainty:}

$$\begin{array}{lccll} \text{maximize} & & t & & \\ \text{subject to} & 0 \le & -t + \sum_{j=1}^{n} (\delta_j) x_j - \Omega z, & & \\ & 0 \le & z - \sqrt{\sum_{j=1}^{n} \sigma_j^2 x_j^2}, & & \\ & & \sum_{j=1}^{n} x_j & = & 1, \\ & 0 \le & x_j, & & j = 1, \dots, n. \end{array} \tag{11.39}$$

Note that the problem (11.39) is essentially the risk-averted portfolio model proposed in mid-50's by Markowitz.

The solution of (11.38) is evident — our '1 should be invested in the asset(s) with the largest possible *guaranteed* return $\delta_j - \sigma_j$. In contrast to this very conservative policy (which in reality prescribes to keep the initial capital in a bank or in the most reliable, and thus low profit, assets), the optimal solution to (11.39) prescribes a quite reasonable diversification of investments which allows to get much better total return than (11.38) with basically zero risk[2]. To illustrate this, assume that there are $n = 300$ assets with the nominal returns (per year) varying from 1.04 (bank savings) to 2.00:

$$\delta_j = 1.04 + 0.96\frac{j-1}{n-1}, \quad j = 1, 2, \ldots, n = 300$$

and volatilities varying from 0 for the bank savings to 1.2 for the most promising asset:

$$\sigma_j = 1.152\frac{j-1}{n-1}, \quad j = 1, \ldots, n = 300.$$

In Listing 11.14 a MATLAB script which builds the associated problem (11.39), solves it via the **MOSEK** optimization toolbox, displays the resulting robust optimal value of the total return and the distribution of investments, and finally runs 10,000 simulations to get the distribution of the total return on the resulting portfolio (in these simulations, the returns on all assets are uniformly distributed in the corresponding intervals) is presented.

Listing 11.14: Script that implements problem (11.39).

```
function rlo2(n, Omega, draw)

n = str2num(n)
Omega = str2num(Omega)
draw

% Set nominal returns and volatilities
delta = (0.96/(n-1))*[0:1:n-1]+1.04;
sigma = (1.152/(n-1))*[0:1:n-1];

% Set mosekopt description of the problem
prob.c = -[1;zeros(2*n+1,1)];
A      = [-1,ones(1,n)+delta,-Omega,zeros(1,n);zeros(n+1,2*n+2)];
for j=1:n,
    % Body of the constraint y(j) - sigma(j)*x(j) = 0:
    A(j+1,j+1)   = -sigma(j);
    A(j+1,2+n+j) = 1;
end;
A(n+2,2:n+1)        = ones(1,n);
prob.a             = sparse(A);
prob.blc           = [zeros(n+1,1);1];
prob.buc           = [inf;zeros(n,1);1];
prob.blx           = [-inf;zeros(n,1);0;zeros(n,1)];
prob.bux           = inf*ones(2*n+2,1);
prob.cones         = cell(1,1);
prob.cones{1}.type = 'MSK_CT_QUAD';
prob.cones{1}.sub  = [n+2;[n+3:1:2*n+2]'];

% Run mosekopt
[r,res]=mosekopt('minimize echo(1)',prob);

if draw == true
    % Display the solution
    xx = res.sol.itr.xx;
    t  = xx(1);

    disp(sprintf('Robust optimal value: %5.4f',t));
    x = max(xx(2:1+n),zeros(n,1));
    plot([1:1:n],x,'-m');
```

(continues on next page)

[2] Recall that in our discussion we have assumed the returns on different assets to be independent of each other. In reality, this is not so and this is why diversification of investments, although reducing the risk, never eliminates it completely

(continued from previous page)

```
    grid on;

    disp('Press <Enter> to run simulations');
    pause

    % Run simulations

    Nsim = 10000;
    out  = zeros(Nsim,1);
    for i=1:Nsim,
        returns  = delta+(2*rand(1,n)-1).*sigma;
        out(i)   = returns*x;
    end;
    disp(sprintf('Actual returns over %d simulations:',Nsim));
    disp(sprintf('Min=%5.4f Mean=%5.4f Max=%5.4f StD=%5.2f',...
                 min(out),mean(out),max(out),std(out)));
    hist(out);
end
```

Here are the results displayed by the script:

Listing 11.15: Output of script *rlo2.m*.

```
Robust optimal value: 1.3428
Actual returns over 10000 simulations:
Min=1.5724 Mean=1.6965 Max=1.8245 StD= 0.03
```

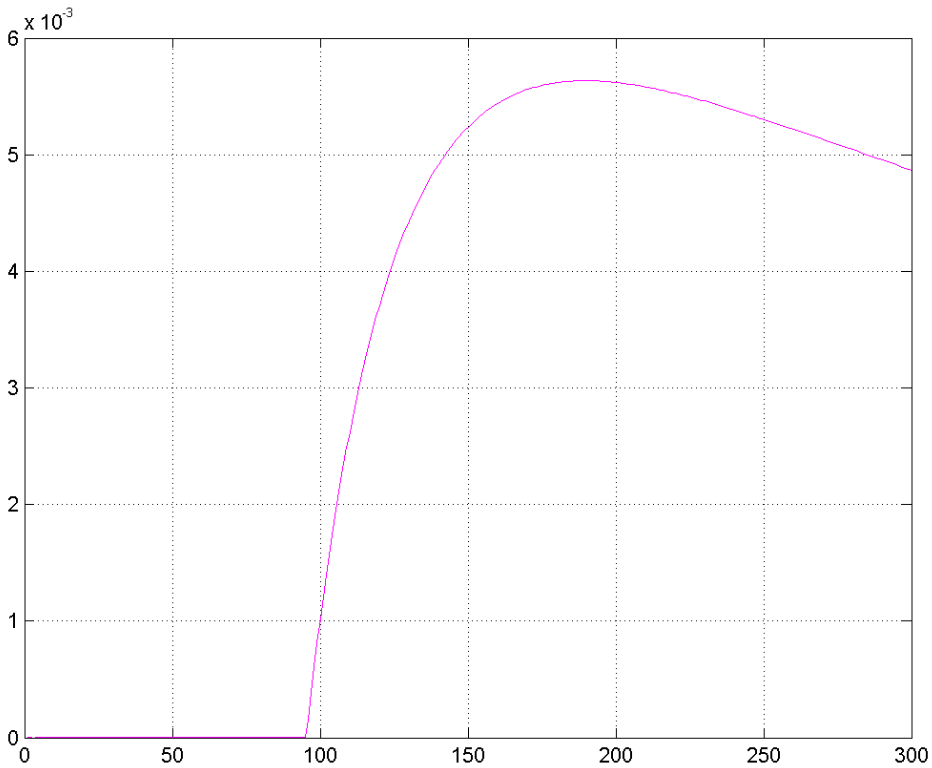

Fig. 11.4: Distribution of investments among the assets in the optimal solution of.

Please note that with our set-up there is exactly one asset with guaranteed return greater than 1 – asset # 1 (bank savings, return 1.04, zero volatility). Consequently, the interval robust counterpart

(11.38) prescribes to put our ‘ #1 in the bank, thus getting a 4% profit. In contrast to this, the diversified portfolio given by the optimal solution of (11.39) never yields profit less than 57.2%, and yields at average a 69.67% profit with pretty low (0.03) standard deviation. We see that in favorable circumstances the ellipsoidal robust counterpart of an uncertain linear program indeed is less conservative than, although basically as reliable as, the interval robust counterpart.

Finally, let us compare our results with those given by the nominal optimal solution. The latter prescribes to invest everything we have in the most promising asset (in our example this is the asset # 300 with a nominal return of 2.00 and volatility of 1.152). Assuming that the actual return is uniformly distributed in the corresponding interval and running 10,000 simulations, we get the following results:

```
Nominal optimal value: 2.0000
Actual returns over 10000 simulations:
Min=0.8483 Mean=1.9918 Max=3.1519 StD= 0.66
```

We see that the nominal solution results in a portfolio which is much more risky, although better at average, than the portfolio given by the robust solution.

Combined Interval-Ellipsoidal Robust Counterpart

We have considered the case when the coefficients a_j of uncertain linear constraint (11.35) are affected by uncorrelated random perturbations symmetrically distributed in given intervals $[-\sigma_j, \sigma_j]$, and we have discussed two ways to model the uncertainty:

- The interval uncertainty model (the uncertainty set $\mathcal{U}$ is the box B), where we ignore the stochastic nature of the perturbations and their independence. This model yields the Interval Robust Counterpart (11.36);
- The ellipsoidal uncertainty model ($\mathcal{U}$ is the ellipsoid $E(\Omega)$), which takes into account the stochastic nature of data perturbations and yields the Ellipsoidal Robust Counterpart (11.37).

Please note that although for large n the ellipsoid $E(\Omega)$ in its diameter, volume and average linear sizes is incomparably smaller than the box B, in the case of $\Omega > 1$ the ellipsoid $E(\Omega)$ in certain directions goes beyond the box. E.g. the ellipsoid $E(6)$, although much more narrow than B in most of the directions, is 6 times wider than B in the directions of the coordinate axes. Intuition says that it hardly makes sense to keep in the uncertainty set realizations of the data which are outside of B and thus forbidden by our model of perturbations, so in the situation under consideration the intersection of $E(\Omega)$ and B is a better model of the uncertainty set than the ellipsoid $E(\Omega)$ itself. What happens when the model of the uncertainty set is the *combined interval-ellipsoidal* uncertainty $\mathcal{U}(\Omega) = E(\Omega) \cap B$?

First, it turns out that the RC of (11.35) corresponding to the uncertainty set $\mathcal{U}(\Omega)$ is still given by a system of linear and conic quadratic inequalities, specifically the system

$$\begin{array}{rcccl} l & \leq & \sum_{j=1}^n a_j^{\mathrm{n}} x_j - \sum_{j=1}^n \sigma_j y_j - \Omega\sqrt{\sum_{j=1}^n \sigma_j^2 u_j^2}, & & \\ & & \sum_{j=1}^n a_j^{\mathrm{n}} x_j + \sum_{j=1}^n \sigma_j z_j + \Omega\sqrt{\sum_{j=1}^n \sigma_j^2 v_j^2} & \leq & u, \\ -y_j & \leq & x_j - u_j & \leq & y_j, j = 1, \ldots, n, \\ -z_j & \leq & x_j - v_j & \leq & z_j, j = 1, \ldots, n. \end{array} \tag{11.40}$$

Second, it turns out that our intuition is correct: As a model of uncertainty, $U(\Omega)$ is as reliable as the ellipsoid $E(\Omega)$. Specifically, if x can be extended to a feasible solution of (11.40), then the probability for x to satisfy a realization of (11.35) is $\geq 1 - \exp\{-\Omega^2/2\}$.

The conclusion is that if we have reasons to assume that the perturbations of uncertain coefficients in a constraint of an uncertain linear optimization problem are (a) random, (b) independent of each other, and (c) symmetrically distributed in given intervals, then it makes sense to associate with this constraint an interval-ellipsoidal model of uncertainty and use a system of linear and conic quadratic inequalities (11.40). Please note that when building the robust counterpart of an uncertain linear optimization problem, one can use different models of the uncertainty (e.g., interval, ellipsoidal, combined interval-ellipsoidal) for different uncertain constraints within the same problem.

Chapter 12

Problem Formulation and Solutions

In this chapter we will discuss the following issues:

- The formal, mathematical formulations of the problem types that **MOSEK** can solve and their duals.
- The solution information produced by **MOSEK**.
- The infeasibility certificate produced by **MOSEK** if the problem is infeasible.

For the underlying mathematical concepts, derivations and proofs see the Modeling Cookbook or any book on convex optimization. This chapter explains how the related data is organized specifically within the **MOSEK** API.

12.1 Linear Optimization

MOSEK accepts linear optimization problems of the form

$$\begin{array}{lccccc} \text{minimize} & & & c^T x + c^f & & \\ \text{subject to} & l^c & \leq & Ax & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x, \end{array} \tag{12.1}$$

where

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

Lower and upper bounds can be infinite, or in other words the corresponding bound may be omitted.

A primal solution (x) is *(primal) feasible* if it satisfies all constraints in (12.1). If (12.1) has at least one primal feasible solution, then (12.1) is said to be (primal) feasible. In case (12.1) does not have a feasible solution, the problem is said to be *(primal) infeasible*

12.1.1 Duality for Linear Optimization

Corresponding to the primal problem (12.1), there is a dual problem

$$
\begin{array}{lccl}
\text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x + c^f & & \\
\text{subject to} & A^T y + s_l^x - s_u^x & = & c, \\
 & -y + s_l^c - s_u^c & = & 0, \\
 & s_l^c, s_u^c, s_l^x, s_u^x & \geq & 0.
\end{array}
\tag{12.2}
$$

If a bound in the primal problem is plus or minus infinity, the corresponding dual variable is fixed at 0, and we use the convention that the product of the bound value and the corresponding dual variable is 0. This is equivalent to removing variable $(s_l^x)_j$ from the dual problem. In other words:

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

A solution

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

to the dual problem is feasible if it satisfies all the constraints in (12.2). If (12.2) has at least one feasible solution, then (12.2) is *(dual) feasible*, otherwise the problem is *(dual) infeasible*.

A solution

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

is denoted a *primal-dual feasible solution*, if (x^*) is a solution to the primal problem (12.1) and $(y^*, (s_l^c)^*, (s_u^c)^*, (s_l^x)^*, (s_u^x)^*)$ is a solution to the corresponding dual problem (12.2). We also define an auxiliary vector

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

containing the activities of linear constraints.

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

$$
\begin{array}{l}
c^T x^* + c^f - \left\{ (l^c)^T (s_l^c)^* - (u^c)^T (s_u^c)^* + (l^x)^T (s_l^x)^* - (u^x)^T (s_u^x)^* + c^f \right\} \\
= \sum_{i=0}^{m-1} \left[(s_l^c)_i^* ((x_i^c)^* - l_i^c) + (s_u^c)_i^* (u_i^c - (x_i^c)^*) \right] \\
+ \sum_{j=0}^{n-1} \left[(s_l^x)_j^* (x_j - l_j^x) + (s_u^x)_j^* (u_j^x - x_j^*) \right] \geq 0
\end{array}
\tag{12.3}
$$

where the first relation can be obtained by transposing and multiplying the dual constraints (12.2) by x^* and $(x^c)^*$ respectively, and the second relation comes from the fact that each term in each sum is nonnegative. It follows that the primal objective will always be greater than or equal to the dual objective.

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

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

are satisfied.

If (12.1) has an optimal solution and **MOSEK** solves the problem successfully, both the primal and dual solution are reported, including a status indicating the exact state of the solution.

12.1.2 Infeasibility for Linear Optimization

Primal Infeasible Problems

If the problem (12.1) is infeasible (has no feasible solution), **MOSEK** will report a certificate of primal infeasibility: The dual solution reported is the certificate of infeasibility, and the primal solution is undefined.

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

$$
\begin{array}{ll}
\text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x \\
\text{subject to} & \\
& A^T y + s_l^x - s_u^x = 0, \\
& -y + s_l^c - s_u^c = 0, \\
& s_l^c, s_u^c, s_l^x, s_u^x \geq 0,
\end{array} \tag{12.4}
$$

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

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

to (12.4) so that

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

Such a solution implies that (12.4) is unbounded, and that (12.1) is infeasible.

Dual Infeasible Problems

If the problem (12.2) is infeasible (has no feasible solution), **MOSEK** will report a certificate of dual infeasibility: The primal solution reported is the certificate of infeasibility, and the dual solution is undefined.

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

$$
\begin{array}{lcccc}
\text{minimize} & & c^T x & & \\
\text{subject to} & \hat{l}^c & \leq & Ax & \leq & \hat{u}^c, \\
& \hat{l}^x & \leq & x & \leq & \hat{u}^x,
\end{array} \tag{12.5}
$$

where

$$
\hat{l}_i^c = \left\{ \begin{array}{ll} 0 & \text{if } l_i^c > -\infty, \\ -\infty & \text{otherwise,} \end{array} \right\} \quad \text{and} \quad \hat{u}_i^c := \left\{ \begin{array}{ll} 0 & \text{if } u_i^c < \infty, \\ \infty & \text{otherwise,} \end{array} \right\}
$$

and

$$
\hat{l}_j^x = \left\{ \begin{array}{ll} 0 & \text{if } l_j^x > -\infty, \\ -\infty & \text{otherwise,} \end{array} \right\} \quad \text{and} \quad \hat{u}_j^x := \left\{ \begin{array}{ll} 0 & \text{if } u_j^x < \infty, \\ \infty & \text{otherwise,} \end{array} \right\}
$$

such that

$$
c^T x < 0.
$$

Such a solution implies that (12.5) is unbounded, and that (12.2) is infeasible.

In case that both the primal problem (12.1) and the dual problem (12.2) are infeasible, **MOSEK** will report only one of the two possible certificates — which one is not defined (**MOSEK** returns the first certificate found).

12.1.3 Minimalization vs. Maximalization

When the objective sense of problem (12.1) is maximization, i.e.

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

the objective sense of the dual problem changes to minimization, and the domain of all dual variables changes sign in comparison to (12.2). The dual problem thus takes the form

$$
\begin{array}{ll}
\text{minimize} & (l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x + c^f \\
\text{subject to} & \\
& A^T y + s_l^x - s_u^x = c, \\
& -y + s_l^c - s_u^c = 0, \\
& s_l^c, s_u^c, s_l^x, s_u^x \leq 0.
\end{array}
$$

This means that the duality gap, defined in (12.3) as the primal minus the dual objective value, becomes nonpositive. It follows that the dual objective will always be greater than or equal to the primal objective. The primal infeasibility certificate will be reported by **MOSEK** as a solution to the system

$$\begin{array}{c} A^T y + s_l^x - s_u^x = 0, \\ -y + s_l^c - s_u^c = 0, \\ s_l^c, s_u^c, s_l^x, s_u^x \leq 0, \end{array} \tag{12.6}$$

such that the objective value is strictly negative

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

Similarly, the certificate of dual infeasibility is an x satisfying the requirements of (12.5) such that $c^T x > 0$.

12.2 Conic Optimization

Conic optimization is an extension of linear optimization (see Sec. 12.1) allowing conic domains to be specified for subsets of the problem variables. A conic optimization problem to be solved by **MOSEK** can be written as

$$\begin{array}{lccccc} \text{minimize} & & & c^T x + c^f & & \\ \text{subject to} & l^c & \leq & Ax & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x, \\ & & & x \in \mathcal{K}, & & \end{array} \tag{12.7}$$

where

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

Lower and upper bounds can be infinite, or in other words the corresponding bound may be omitted.

The set $\mathcal{K}$ is a Cartesian product of convex cones, namely $\mathcal{K} = \mathcal{K}_1 \times \cdots \times \mathcal{K}_p$. Having the domain restriction $x \in \mathcal{K}$, is thus equivalent to

$$x^t \in \mathcal{K}_t \subseteq \mathbb{R}^{n_t},$$

where $x = (x^1, \ldots, x^p)$ is a partition of the problem variables. Please note that the n-dimensional Euclidean space $\mathbb{R}^n$ is a cone itself, so simple linear variables are still allowed. The user only needs to specify subsets of variables which belong to non-trivial cones.

In this section we discuss the formulations which apply to the following cones supported by **MOSEK**:

- The set $\mathbb{R}^n$.
- The zero cone $\{(0, \ldots, 0)\}$.
- Quadratic cone

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

- Rotated quadratic cone

$$\mathcal{Q}_r^n = \left\{ x \in \mathbb{R}^n : 2x_1 x_2 \geq \sum_{j=3}^{n} x_j^2, \quad x_1 \geq 0, \quad x_2 \geq 0 \right\}.$$

- Primal exponential cone

$$K_{\exp} = \left\{ x \in \mathbb{R}^3 : x_1 \geq x_2 \exp(x_3/x_2), \quad x_1, x_2 \geq 0 \right\}$$

as well as its dual

$$K_{\exp}^* = \left\{ x \in \mathbb{R}^3 : x_1 \geq -x_3 e^{-1} \exp(x_2/x_3), \quad x_3 \leq 0, x_1 \geq 0 \right\}.$$

- Primal power cone (with parameter $0 < \alpha < 1$)

$$\mathcal{P}_n^{\alpha,1-\alpha} = \left\{ x \in \mathbb{R}^n : x_1^\alpha x_2^{1-\alpha} \geq \sqrt{\sum_{j=3}^{n} x_j^2}, \quad x_1, x_2 \geq 0 \right\}$$

as well as its dual

$$(\mathcal{P}_n^{\alpha,1-\alpha})^* = \left\{ x \in \mathbb{R}^n : \left(\frac{x_1}{\alpha}\right)^\alpha \left(\frac{x_2}{1-\alpha}\right)^{1-\alpha} \geq \sqrt{\sum_{j=3}^{n} x_j^2}, \quad x_1, x_2 \geq 0 \right\}.$$

MOSEK supports also the cone of positive semidefinite matrices. Since that is handled through a separate interface, we discuss it in Sec. 12.3.

12.2.1 Duality for Conic Optimization

Corresponding to the primal problem (12.7), there is a dual problem

$$\begin{array}{ll} \text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x + c^f \\ \text{subject to} & \\ & A^T y + s_l^x - s_u^x + s_n^x = c \\ & -y + s_l^c - s_u^c = 0, \\ & s_l^c, s_u^c, s_l^x, s_u^x \geq 0, \\ & s_n^x \in \mathcal{K}^*, \end{array} \tag{12.8}$$

where the dual cone $\mathcal{K}^*$ is a Cartesian product of the cones dual to $\mathcal{K}_t$. In practice this means that s_n^x has one entry for each entry in x. Please note that the dual problem of the dual problem is identical to the original primal problem.

If a bound in the primal problem is plus or minus infinity, the corresponding dual variable is fixed at 0, and we use the convention that the product of the bound value and the corresponding dual variable is 0. This is equivalent to removing variable $(s_l^x)_j$ from the dual problem. In other words:

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

A solution

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

to the dual problem is feasible if it satisfies all the constraints in (12.8). If (12.8) has at least one feasible solution, then (12.8) is *(dual) feasible*, otherwise the problem is *(dual) infeasible*.

A solution

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

is denoted a *primal-dual feasible solution*, if (x^*) is a solution to the primal problem (12.7) and $(y^*, (s_l^c)^*, (s_u^c)^*, (s_l^x)^*, (s_u^x)^*, (s_n^x)^*)$ is a solution to the corresponding dual problem (12.8). We also define an auxiliary vector

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

containing the activities of linear constraints.

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

$$\begin{array}{l} c^T x^* + c^f - \left\{(l^c)^T (s_l^c)^* - (u^c)^T (s_u^c)^* + (l^x)^T (s_l^x)^* - (u^x)^T (s_u^x)^* + c^f\right\} \\ = \sum_{i=0}^{m-1} \left[(s_l^c)_i^* ((x_i^c)^* - l_i^c) + (s_u^c)_i^* (u_i^c - (x_i^c)^*)\right] \\ + \sum_{j=0}^{n-1} \left[(s_l^x)_j^* (x_j - l_j^x) + (s_u^x)_j^* (u_j^x - x_j^*)\right] + \sum_{j=0}^{n-1} (s_n^x)_j^* x_j^* \geq 0 \end{array} \tag{12.9}$$

where the first relation can be obtained by transposing and multiplying the dual constraints (12.2) by x^* and $(x^c)^*$ respectively, and the second relation comes from the fact that each term in each sum is nonnegative. It follows that the primal objective will always be greater than or equal to the dual objective.

It is well-known that, under some non-degeneracy assumptions that exclude ill-posed cases, a conic optimization problem has an optimal solution if and only if there exist feasible primal-dual solution so that the duality gap is zero, or, equivalently, that the *complementarity conditions*

$$\begin{array}{rcll} (s_l^c)_i^* ((x_i^c)^* - l_i^c) & = & 0, & i = 0, \ldots, m-1, \\ (s_u^c)_i^* (u_i^c - (x_i^c)^*) & = & 0, & i = 0, \ldots, m-1, \\ (s_l^x)_j^* (x_j^* - l_j^x) & = & 0, & j = 0, \ldots, n-1, \\ (s_u^x)_j^* (u_j^x - x_j^*) & = & 0, & j = 0, \ldots, n-1, \\ \sum_{j=0}^{n-1} (s_n^x)_j^* x_j^* & = & 0. & \end{array} \tag{12.10}$$

are satisfied.

If (12.7) has an optimal solution and **MOSEK** solves the problem successfully, both the primal and dual solution are reported, including a status indicating the exact state of the solution.

12.2.2 Infeasibility for Conic Optimization

Primal Infeasible Problems

If the problem (12.7) is infeasible (has no feasible solution), **MOSEK** will report a certificate of primal infeasibility: The dual solution reported is the certificate of infeasibility, and the primal solution is undefined.

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

$$\begin{array}{ll} \text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x \\ \text{subject to} & \\ & A^T y + s_l^x - s_u^x + s_n^x = 0, \\ & -y + s_l^c - s_u^c = 0, \\ & s_l^c, s_u^c, s_l^x, s_u^x \geq 0, \\ & s_n^x \in \mathcal{K}^*, \end{array} \tag{12.11}$$

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

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

to (12.11) so that

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

Such a solution implies that (12.11) is unbounded, and that (12.7) is infeasible.

Dual Infeasible Problems

If the problem (12.8) is infeasible (has no feasible solution), **MOSEK** will report a certificate of dual infeasibility: The primal solution reported is the certificate of infeasibility, and the dual solution is undefined.

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

$$
\begin{array}{lccccc}
\text{minimize} & & & c^T x & & \\
\text{subject to} & \hat{l}^c & \leq & Ax & \leq & \hat{u}^c, \\
 & \hat{l}^x & \leq & x & \leq & \hat{u}^x, \\
 & & & x \in K, & &
\end{array}
\tag{12.12}
$$

where

$$
\hat{l}_i^c = \left\{ \begin{array}{ll} 0 & \text{if } l_i^c > -\infty, \\ -\infty & \text{otherwise,} \end{array} \right\} \quad \text{and} \quad \hat{u}_i^c := \left\{ \begin{array}{ll} 0 & \text{if } u_i^c < \infty, \\ \infty & \text{otherwise,} \end{array} \right\}
\tag{12.13}
$$

and

$$
\hat{l}_j^x = \left\{ \begin{array}{ll} 0 & \text{if } l_j^x > -\infty, \\ -\infty & \text{otherwise,} \end{array} \right\} \quad \text{and} \quad \hat{u}_j^x := \left\{ \begin{array}{ll} 0 & \text{if } u_j^x < \infty, \\ \infty & \text{otherwise,} \end{array} \right\}
\tag{12.14}
$$

such that

$$
c^T x < 0.
$$

Such a solution implies that (12.12) is unbounded, and that (12.8) is infeasible.

In case that both the primal problem (12.7) and the dual problem (12.8) are infeasible, **MOSEK** will report only one of the two possible certificates — which one is not defined (**MOSEK** returns the first certificate found).

12.2.3 Minimalization vs. Maximalization

When the objective sense of problem (12.7) is maximization, i.e.

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

the objective sense of the dual problem changes to minimization, and the domain of all dual variables changes sign in comparison to (12.2). The dual problem thus takes the form

$$
\begin{array}{ll}
\text{minimize} & (l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x + c^f \\
\text{subject to} & A^T y + s_l^x - s_u^x + s_n^x = c, \\
 & -y + s_l^c - s_u^c = 0, \\
 & s_l^c, s_u^c, s_l^x, s_u^x \leq 0, \\
 & -s_n^x \in \mathcal{K}^*
\end{array}
$$

This means that the duality gap, defined in (12.9) as the primal minus the dual objective value, becomes nonpositive. It follows that the dual objective will always be greater than or equal to the primal objective. The primal infeasibility certificate will be reported by **MOSEK** as a solution to the system

$$
\begin{array}{c}
A^T y + s_l^x - s_u^x + s_n^x = 0, \\
-y + s_l^c - s_u^c = 0, \\
s_l^c, s_u^c, s_l^x, s_u^x \leq 0, \\
-s_n^x \in \mathcal{K}^*
\end{array}
\tag{12.15}
$$

such that the objective value is strictly negative

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

Similarly, the certificate of dual infeasibility is an x satisfying the requirements of (12.12) such that $c^T x > 0$.

12.3 Semidefinite Optimization

Semidefinite optimization is an extension of conic optimization (see Sec. 12.2) allowing positive semidefinite matrix variables to be used in addition to the usual scalar variables. All the other parts of the input are defined exactly as in Sec. 12.2, and the discussion from that section applies verbatim to all properties of problems with semidefinite variables. We only briefly indicate how the corresponding formulae should be modified with semidefinite terms.

A semidefinite optimization problem can be written as

$$
\begin{array}{lccccl}
\text{minimize} & & & \sum_{j=0}^{n-1} c_j x_j + \sum_{j=0}^{p-1} \left\langle \overline{C}_j, \overline{X}_j \right\rangle + c^f & & \\
\text{subject to} & l_i^c & \leq & \sum_{j=0}^{n-1} a_{ij} x_j + \sum_{j=0}^{p-1} \left\langle \overline{A}_{ij}, \overline{X}_j \right\rangle & \leq & u_i^c, \quad i = 0, \ldots, m-1 \\
& l_j^x & \leq & x_j & \leq & u_j^x, \quad j = 0, \ldots, n-1 \\
& & & x \in \mathcal{K}, & & \\
& & & \overline{X}_j \in \mathcal{S}_+^{r_j}, & & \quad j = 0, \ldots, p-1
\end{array}
\tag{12.16}
$$

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

$$
\langle U, V \rangle := \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} U_{ij} V_{ij}.
$$

As always we write $A = (a_{i,j})$ for the linear coefficient matrix.

Duality

The definition of the dual problem (12.8) becomes:

$$
\begin{array}{ll}
\text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x + c^f \\
\text{subject to} & \\
& A^T y + s_l^x - s_u^x + s_n^x = c \\
& -y + s_l^c - s_u^c = 0, \\
& \overline{C}_j - \sum_{i=0}^{m-1} y_i \overline{A}_{ij} = \overline{S}_j, \qquad j = 0, \ldots, p-1 \\
& s_l^c, s_u^c, s_l^x, s_u^x \geq 0, \\
& s_n^x \in \mathcal{K}^*, \\
& \overline{S}_j \in \mathcal{S}_+^{r_j}, \qquad j = 0, \ldots, p-1.
\end{array}
\tag{12.17}
$$

The duality gap (12.9) is computed as:

$$
\begin{array}{l}
c^T x^* + c^f - \left\{ (l^c)^T (s_l^c)^* - (u^c)^T (s_u^c)^* + (l^x)^T (s_l^x)^* - (u^x)^T (s_u^x)^* + c^f \right\} \\
= \sum_{i=0}^{m-1} \left[(s_l^c)_i^* ((x_i^c)^* - l_i^c) + (s_u^c)_i^* (u_i^c - (x_i^c)^*) \right] \\
+ \sum_{j=0}^{n-1} \left[(s_l^x)_j^* (x_j - l_j^x) + (s_u^x)_j^* (u_j^x - x_j^*) \right] + \sum_{j=0}^{n-1} (s_n^x)_j^* x_j^* + \sum_{j=0}^{p-1} \langle \overline{X}_j, \overline{S}_j \rangle \geq 0.
\end{array}
\tag{12.18}
$$

Complementarity conditions (12.10) include the additional relation:

$$
\langle \overline{X}_j, \overline{S}_j \rangle = 0 \quad j = 0, \ldots, p-1. \tag{12.19}
$$

Infeasibility

A certificate of primal infeasibility (12.11) is now a feasible solution to:

$$
\begin{array}{ll}
\text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x \\
\text{subject to} & \\
& A^T y + s_l^x - s_u^x + s_n^x = 0, \\
& -y + s_l^c - s_u^c = 0, \\
& \sum_{i=0}^{m-1} y_i \overline{A}_{ij} + \overline{S}_j = 0, \qquad j = 0, \ldots, p-1 \\
& s_l^c, s_u^c, s_l^x, s_u^x \geq 0, \\
& s_n^x \in \mathcal{K}^*, \\
& \overline{S}_j \in \mathcal{S}_+^{r_j}, \qquad j = 0, \ldots, p-1.
\end{array}
\tag{12.20}
$$

such that the objective value is strictly positive.

Similarly, a dual infeasibility certificate (12.12) is a feasible solution to

$$\begin{array}{lccccl}
\text{minimize} & & \sum_{j=0}^{n-1} c_j x_j + \sum_{j=0}^{p-1} \langle \overline{C}_j, \overline{X}_j \rangle & & & \\
\text{subject to} & \hat{l}_i^c \leq & \sum_{j=0}^{n-1} a_{ij} x_j + \sum_{j=0}^{p-1} \langle \overline{A}_{ij}, \overline{X}_j \rangle & \leq & \hat{u}_i^c, & i = 0, \ldots, m-1 \\
& \hat{l}_j^x \leq & x_j & \leq & \hat{u}_j^x, & j = 0, \ldots, n-1 \\
& & x \in \mathcal{K}, & & & \\
& & \overline{X}_j \in \mathcal{S}_+^{r_j}, & & & j = 0, \ldots, p-1
\end{array} \tag{12.21}$$

where the modified bounds are as in (12.13) and (12.14) and the objective value is strictly negative.

12.4 Quadratic and Quadratically Constrained Optimization

A convex quadratic and quadratically constrained optimization problem has the form

$$\begin{array}{lccccl}
\text{minimize} & & \frac{1}{2} x^T Q^o x + c^T x + c^f & & & \\
\text{subject to} & l_k^c \leq & \frac{1}{2} x^T Q^k x + \sum_{j=0}^{n-1} a_{kj} x_j & \leq & u_k^c, & k = 0, \ldots, m-1, \\
& l_j^x \leq & x_j & \leq & u_j^x, & j = 0, \ldots, n-1,
\end{array} \tag{12.22}$$

where all variables and bounds have the same meaning as for linear problems (see Sec. 12.1) and Q^o and all Q^k are symmetric matrices. Moreover, for convexity, Q^o must be a positive semidefinite matrix and Q^k must satisfy

$$\begin{array}{rcl}
-\infty < l_k^c & \Rightarrow & Q^k \text{ is negative semidefinite}, \\
u_k^c < \infty & \Rightarrow & Q^k \text{ is positive semidefinite}, \\
-\infty < l_k^c \leq u_k^c < \infty & \Rightarrow & Q^k = 0.
\end{array}$$

The convexity requirement is very important and **MOSEK** checks whether it is fulfilled.

12.4.1 A Recommendation

Any convex quadratic optimization problem can be reformulated as a conic quadratic optimization problem, see Modeling Cookbook and *[And13]*. In fact **MOSEK** does such conversion internally as a part of the solution process for the following reasons:

- the conic optimizer is numerically more robust than the one for quadratic problems.
- the conic optimizer is usually faster because quadratic cones are simpler than quadratic functions, even though the conic reformulation usually has more constraints and variables than the original quadratic formulation.
- it is easy to dualize the conic formulation if deemed worthwhile potentially leading to (huge) computational savings.

However, instead of relying on the automatic reformulation we recommend to formulate the problem as a conic problem from scratch because:

- it saves the computational overhead of the reformulation including the convexity check. A conic problem is convex by construction and hence no convexity check is needed for conic problems.
- usually the modeler can do a better reformulation than the automatic method because the modeler can exploit the knowledge of the problem at hand.

To summarize we recommend to formulate quadratic problems and in particular quadratically constrained problems directly in conic form.

12.4.2 Duality for Quadratic and Quadratically Constrained Optimization

The dual problem corresponding to the quadratic and quadratically constrained optimization problem (12.22) is given by

$$
\begin{array}{ll}
\text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x + \frac{1}{2} x^T \left\{ \sum_{k=0}^{m-1} y_k Q^k - Q^o \right\} x + c^f \\
\text{subject to} & A^T y + s_l^x - s_u^x + \left\{ \sum_{k=0}^{m-1} y_k Q^k - Q^o \right\} x = c, \\
& -y + s_l^c - s_u^c = 0, \\
& s_l^c, s_u^c, s_l^x, s_u^x \geq 0.
\end{array}
\tag{12.23}
$$

The dual problem is related to the dual problem for linear optimization (see Sec. 12.1.1), but depends on the variable x which in general can not be eliminated. In the solutions reported by **MOSEK**, the value of x is the same for the primal problem (12.22) and the dual problem (12.23).

12.4.3 Infeasibility for Quadratic Optimization

In case **MOSEK** finds a problem to be infeasible it reports a certificate of infeasibility. We write them out explicitly for quadratic problems, that is when $Q^k = 0$ for all k and quadratic terms appear only in the objective Q^o. In this case the constraints both in the primal and dual problem are linear, and **MOSEK** produces for them the same infeasibility certificate as for linear problems.

The certificate of primal infeasibility is a solution to the problem (12.4) such that the objective value is strictly positive.

The certificate of dual infeasibility is a solution to the problem (12.5) together with an additional constraint

$$Q^o x = 0$$

such that the objective value is strictly negative.

12.5 Affine Conic Constraints

Optimization Toolbox for MATLAB allows conic problems to be specified in another format, namely with *affine conic constraints*. A conic problem can thus be specified as:

$$
\begin{array}{lcccll}
\text{minimize} & & \sum_{j=1}^{n} c_j x_j + \sum_{j=1}^{p} \left\langle \overline{C}_j, \overline{X}_j \right\rangle + c^f & & & \\
\text{subject to} & l_i^c \leq & \sum_{j=1}^{n} a_{ij} x_j + \sum_{j=1}^{p} \left\langle \overline{A}_{ij}, \overline{X}_j \right\rangle & \leq & u_i^c, & i = 1, \ldots, m, \\
& l_j^x \leq & x_j & \leq & u_j^x, & j = 1, \ldots, n, \\
& & Fx + g \in \mathcal{K}, & & & \\
& & \overline{X}_j \in \mathcal{S}_+^{r_j}, & & & j = 1, \ldots, p
\end{array}
\tag{12.24}
$$

where all the data has the same meaning as in Sec. 12.2 and Sec. 12.3 and $F \in \mathbb{R}^{k \times n}$, $g \in \mathbb{R}^k$ specify the affine conic constraint.

Duality

The dual of problem (12.24) is

$$
\begin{array}{lll}
\text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x - \dot{y}^T g + c^f & \\
\text{subject to} & & \\
& A^T y + s_l^x - s_u^x + F^T \dot{y} = c, & \\
& -y + s_l^c - s_u^c = 0, & \\
& \overline{C}_j - \sum_{i=0}^{m-1} y_i \overline{A}_{ij} = \overline{S}_j, & j = 0, \ldots, p-1 \\
& s_l^c, s_u^c, s_l^x, s_u^x \geq 0, & \\
& \dot{y} \in \mathcal{K}^*, & \\
& \overline{S}_j \in \mathcal{S}_+^{r_j}, & j = 0, \ldots, p-1.
\end{array}
\tag{12.25}
$$

Duality and infeasibility cerfificates behave analogously as in Sec. 12.2.

Remark

A problem of this form is internally converted into the problem:

$$
\begin{array}{lrccll}
\text{minimize} & & \sum_{j=1}^{n} c_j x_j + \sum_{j=1}^{p} \left\langle \overline{C}_j, \overline{X}_j \right\rangle + c^f & & & \\
\text{subject to} & l_i^c \leq & \sum_{j=1}^{n} a_{ij} x_j + \sum_{j=1}^{p} \left\langle \overline{A}_{ij}, \overline{X}_j \right\rangle & \leq & u_i^c, & i = 1, \ldots, m, \\
& g_i \leq & z_i - \sum_{j=1}^{n} f_{ij} x_j & \leq & g_i & i = 1, \ldots, k, \\
& l_j^x \leq & x_j & \leq & u_j^x, & j = 1, \ldots, n, \\
& & z \in \mathcal{K}, & & & \\
& & \overline{X}_j \in \mathcal{S}_+^{r_j}, & & & j = 1, \ldots, p
\end{array}
\tag{12.26}
$$

which conforms with the format in Sec. 12.2 and Sec. 12.3. The reformulated problem has $n+k$ variables, $m+k$ linear constraints and in total k variables in cones. The new columns and rows are appended at the end of the original ones. If a problem with affine conic constraints is saved to a file then this reformulation will be written.

Chapter 13

Optimizers

The most essential part of **MOSEK** are the optimizers:

- *primal simplex* (linear problems),
- *dual simplex* (linear problems),
- *interior-point* (linear, quadratic and conic problems),
- *mixed-integer* (problems with integer variables).

The structure of a successful optimization process is roughly:

- **Presolve**
  1. *Elimination*: Reduce the size of the problem.
  2. *Dualizer*: Choose whether to solve the primal or the dual form of the problem.
  3. *Scaling*: Scale the problem for better numerical stability.
- **Optimization**
  1. *Optimize*: Solve the problem using selected method.
  2. *Terminate*: Stop the optimization when specific termination criteria have been met.
  3. *Report*: Return the solution or an infeasibility certificate.

The preprocessing stage is transparent to the user, but useful to know about for tuning purposes. The purpose of the preprocessing steps is to make the actual optimization more efficient and robust. We discuss the details of the above steps in the following sections.

13.1 Presolve

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

1. remove redundant constraints,
2. eliminate fixed variables,
3. remove linear dependencies,
4. substitute out (implied) free variables, and
5. reduce the size of the optimization problem in general.

After the presolved problem has been optimized the solution is automatically postsolved so that the returned solution is valid for the original problem. Hence, the presolve is completely transparent. For further details about the presolve phase, please see *[AA95]* and *[AGMX96]*.

It is possible to fine-tune the behavior of the presolve or to turn it off entirely. If presolve consumes too much time or memory compared to the reduction in problem size gained it may be disabled. This is done by setting the parameter `MSK_IPAR_PRESOLVE_USE` to `"MSK_PRESOLVE_MODE_OFF"`. The two most time-consuming steps of the presolve are

- the eliminator, and
- the linear dependency check.

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

Numerical issues in the presolve

During the presolve the problem is reformulated so that it hopefully solves faster. However, in rare cases the presolved problem may be harder to solve then the original problem. The presolve may also be infeasible although the original problem is not. If it is suspected that presolved problem is much harder to solve than the original, we suggest to first turn the eliminator off by setting the parameter *MSK_IPAR_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 *MSK_DPAR_PRESOLVE_TOL_X* and *MSK_DPAR_PRESOLVE_TOL_S*. However, if reducing the parameters actually helps then this should be taken as an indication that the problem is badly formulated.

Eliminator

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

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

y is an implied free variable that can be substituted out of the problem, if deemed worthwhile. If the eliminator consumes too much time or memory compared to the reduction in problem size gained it may be disabled. This can be done by setting the parameter *MSK_IPAR_PRESOLVE_ELIMINATOR_MAX_NUM_TRIES* to 0. In rare cases the eliminator may cause that the problem becomes much hard to solve.

Linear dependency checker

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

$$\begin{array}{rcl} x_1 + x_2 + x_3 & = & 1, \\ x_1 + 0.5x_2 & = & 0.5, \\ 0.5x_2 + x_3 & = & 0.5. \end{array}$$

contain exactly one linear dependency. This implies that one of the constraints can be dropped without changing the set of feasible solutions. Removing linear dependencies is in general a good idea since it reduces the size of the problem. Moreover, the linear dependencies are likely to introduce numerical problems in the optimization phase. It is best practice to build models without linear dependencies, but that is not always easy for the user to control. If the linear dependencies are removed at the modeling stage, the linear dependency check can safely be disabled by setting the parameter *MSK_IPAR_PRESOLVE_LINDEP_USE* to *"MSK_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:

- *MSK_IPAR_INTPNT_SOLVE_FORM*: In case of the interior-point optimizer.
- *MSK_IPAR_SIM_SOLVE_FORM*: In case of the simplex optimizer.

Note that currently only linear and conic (but not semidefinite) problems may be automatically dualized.

Scaling

Problems containing data with large and/or small coefficients, say $1.0e+9$ or $1.0e-7$, are often hard to solve. Significant digits may be truncated in calculations with finite precision, which can result in the optimizer relying on inaccurate data. Since computers work in finite precision, extreme coefficients should be avoided. In general, data around the same *order of magnitude* is preferred, and we will refer to a problem, satisfying this loose property, as being *well-scaled*. If the problem is not well scaled, **MOSEK** will try to scale (multiply) constraints and variables by suitable constants. **MOSEK** solves the scaled problem to improve the numerical properties.

The scaling process is transparent, i.e. the solution to the original problem is reported. It is important to be aware that the optimizer terminates when the termination criterion is met on the scaled problem, therefore significant primal or dual infeasibilities may occur after unscaling for badly scaled problems. The best solution of this issue is to reformulate the problem, making it better scaled.

By default **MOSEK** heuristically chooses a suitable scaling. The scaling for interior-point and simplex optimizers can be controlled with the parameters `MSK_IPAR_INTPNT_SCALING` and `MSK_IPAR_SIM_SCALING` respectively.

13.2 Linear Optimization

13.2.1 Optimizer Selection

Two different types of optimizers are available for linear problems: The default is an interior-point method, and the alternative is the simplex method (primal or dual). The optimizer can be selected using the parameter `MSK_IPAR_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 `MSK_IPAR_OPTIMIZER` parameter to `"MSK_OPTIMIZER_FREE_SIMPLEX"` instructs **MOSEK** to choose one of the simplex variants automatically.

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

13.2.2 The Interior-point Optimizer

The purpose of this section is to provide information about the algorithm employed in the **MOSEK** interior-point optimizer for linear problems and about its termination criteria.

The homogeneous primal-dual problem

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

$$\begin{array}{lccc} \text{minimize} & c^T x & & \\ \text{subject to} & Ax & = & b, \\ & x \geq 0. & & \end{array} \tag{13.1}$$

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

Since it is not known beforehand whether problem (13.1) has an optimal solution, is primal infeasible or is dual infeasible, the optimization algorithm must deal with all three situations. This is the reason why **MOSEK** solves the so-called homogeneous model

$$\begin{array}{rcl} Ax - b\tau & = & 0, \\ A^T y + s - c\tau & = & 0, \\ -c^T x + b^T y - \kappa & = & 0, \\ x, s, \tau, \kappa & \geq & 0, \end{array} \tag{13.2}$$

where y and s correspond to the dual variables in (13.1), and τ and κ are two additional scalar variables. Note that the homogeneous model (13.2) always has solution since

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

is a solution, although not a very interesting one. Any solution

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

to the homogeneous model (13.2) satisfies

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

Moreover, there is always a solution that has the property $\tau^* + \kappa^* > 0$.

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

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

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

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

is a primal-dual optimal solution (see Sec. 12.1 for the mathematical background on duality and optimality).

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

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

This implies that at least one of

$$c^T x^* < 0 \tag{13.3}$$

or

$$b^T y^* > 0 \tag{13.4}$$

is satisfied. If (13.3) is satisfied then x^* is a certificate of dual infeasibility, whereas if (13.4) is satisfied then y^* is a certificate of primal infeasibility.

In summary, by computing an appropriate solution to the homogeneous model, all information required for a solution to the original problem is obtained. A solution to the homogeneous model can be computed using a primal-dual interior-point algorithm *[And09]*.

Interior-point Termination Criterion

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

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

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

to homogeneous model is generated, where

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

Optimal case

Whenever the trial solution satisfies the criterion

$$\begin{array}{rcl} \left\| A\frac{x^k}{\tau^k} - b \right\|_\infty & \leq & \epsilon_p(1 + \|b\|_\infty), \\ \left\| A^T\frac{y^k}{\tau^k} + \frac{s^k}{\tau^k} - c \right\|_\infty & \leq & \epsilon_d(1 + \|c\|_\infty), \text{ and} \\ \min\left(\frac{(x^k)^T s^k}{(\tau^k)^2}, \left|\frac{c^T x^k}{\tau^k} - \frac{b^T y^k}{\tau^k}\right| \right) & \leq & \epsilon_g \max\left(1, \frac{\min(|c^T x^k|, |b^T y^k|)}{\tau^k}\right), \end{array} \tag{13.5}$$

the interior-point optimizer is terminated and

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

is reported as the primal-dual optimal solution. The interpretation of (13.5) is that the optimizer is terminated if

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

Dual infeasibility certificate

On the other hand, if the trial solution satisfies

$$-\epsilon_i c^T x^k > \frac{\|c\|_\infty}{\max(1, \|b\|_\infty)} \left\| Ax^k \right\|_\infty$$

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

$$\left\| Ax^k \right\|_\infty > 0,$$

and define

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

It is easy to verify that

$$\|A\bar{x}\|_\infty = \epsilon_i \frac{\max(1, \|b\|_\infty)}{\|c\|_\infty} \text{ and } -c^T\bar{x} > 1,$$

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

Primal infeasibility certificate

Finally, if

$$\epsilon_i b^T y^k > \frac{\|b\|_\infty}{\max(1, \|c\|_\infty)} \left\|A^T y^k + s^k\right\|_\infty$$

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

Adjusting optimality criteria

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

Table 13.1: Parameters employed in termination criterion

ToleranceParameter	name
ε_p	*MSK_DPAR_INTPNT_TOL_PFEAS*
ε_d	*MSK_DPAR_INTPNT_TOL_DFEAS*
ε_g	*MSK_DPAR_INTPNT_TOL_REL_GAP*
ε_i	*MSK_DPAR_INTPNT_TOL_INFEAS*

The default values of the termination tolerances are chosen such that for a majority of problems appearing in practice it is not possible to achieve much better accuracy. Therefore, tightening the tolerances usually is not worthwhile. However, an inspection of (13.5) reveals that the quality of the solution depends on $\|b\|_\infty$ and $\|c\|_\infty$; the smaller the norms are, the better the solution accuracy.

The interior-point method as implemented by **MOSEK** will converge toward optimality and primal and dual feasibility at the same rate *[And09]*. This means that if the optimizer is stopped prematurely then it is very unlikely that either the primal or dual solution is feasible. Another consequence is that in most cases all the tolerances, ε_p, ε_d, ε_g and ε_i, have to be relaxed together to achieve an effect.

If the optimizer terminates without locating a solution that satisfies the termination criteria, for example because of a stall or other numerical issues, then it will check if the solution found up to that point satisfies the same criteria with all tolerances multiplied by the value of *MSK_DPAR_INTPNT_CO_TOL_NEAR_REL* . If this is the case, the solution is still declared as optimal.

The basis identification discussed in Sec. 13.2.2 requires an optimal solution to work well; hence basis identification should be turned off if the termination criterion is relaxed.

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

Basis Identification

An interior-point optimizer does not return an optimal basic solution unless the problem has a unique primal and dual optimal solution. Therefore, the interior-point optimizer has an optional post-processing step that computes an optimal basic solution starting from the optimal interior-point solution. More information about the basis identification procedure may be found in *[AY96]*. In the following we provide an overall idea of the procedure.

There are some cases in which a basic solution could be more valuable:

- a basic solution is often more accurate than an interior-point solution,
- a basic solution can be used to warm-start the simplex algorithm in case of reoptimization,
- a basic solution is in general more sparse, i.e. more variables are fixed to zero. This is particularly appealing when solving continuous relaxations of mixed integer problems, as well as in all applications in which sparser solutions are preferred.

To illustrate how the basis identification routine works, we use the following trivial example:

$$\begin{array}{lccc} \text{minimize} & x+y & & \\ \text{subject to} & x+y & = & 1, \\ & x, y \geq 0. & & \end{array}$$

It is easy to see that all feasible solutions are also optimal. In particular, there are two basic solutions, namely

$$\begin{array}{rcl} (x_1^*, y_1^*) & = & (1,0), \\ (x_2^*, y_2^*) & = & (0,1). \end{array}$$

The interior point algorithm will actually converge to the center of the optimal set, i.e. to $(x^*, y^*) = (1/2, 1/2)$ (to see this in **MOSEK** deactivate *Presolve*).

In practice, when the algorithm gets close to the optimal solution, it is possible to construct in polynomial time an initial basis for the simplex algorithm from the current interior point solution. This basis is used to warm-start the simplex algorithm that will provide the optimal basic solution. In most cases the constructed basis is optimal, or very few iterations are required by the simplex algorithm to make it optimal and hence the final *clean-up* phase be short. However, for some cases of ill-conditioned problems the additional simplex clean up phase may take of lot a time.

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

- *MSK_IPAR_INTPNT_BASIS*,
- *MSK_IPAR_BI_IGNORE_MAX_ITER*, and
- *MSK_IPAR_BI_IGNORE_NUM_ERROR*

control when basis identification is performed.

The type of simplex algorithm to be used (primal/dual) can be tuned with the parameter *MSK_IPAR_BI_CLEAN_OPTIMIZER*, and the maximum number of iterations can be set with *MSK_IPAR_BI_MAX_ITERATIONS*.

Finally, it should be mentioned that there is no guarantee on which basic solution will be returned.

The Interior-point Log

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

```
Optimizer  - threads                : 1
Optimizer  - solved problem         : the dual
Optimizer  - Constraints            : 2
Optimizer  - Cones                  : 0
Optimizer  - Scalar variables       : 6                 conic                  : 0
Optimizer  - Semi-definite variables: 0                 scalarized             : 0
Factor     - setup time             : 0.00              dense det. time        : 0.00
Factor     - ML order time          : 0.00              GP order time          : 0.00
Factor     - nonzeros before factor : 3                 after factor           : 3
Factor     - dense dim.             : 0                 flops                  : 7.00e+001
ITE PFEAS    DFEAS    GFEAS    PRSTATUS   POBJ              DOBJ              MU       TIME
0   1.0e+000 8.6e+000 6.1e+000 1.00e+000  0.000000000e+000  -2.208000000e+003 1.0e+000 0.00
1   1.1e+000 2.5e+000 1.6e-001 0.00e+000  -7.901380925e+003 -7.394611417e+003 2.5e+000 0.00
2   1.4e-001 3.4e-001 2.1e-002 8.36e-001  -8.113031650e+003 -8.055866001e+003 3.3e-001 0.00
3   2.4e-002 5.8e-002 3.6e-003 1.27e+000  -7.777530698e+003 -7.766471080e+003 5.7e-002 0.01
4   1.3e-004 3.2e-004 2.0e-005 1.08e+000  -7.668323435e+003 -7.668207177e+003 3.2e-004 0.01
5   1.3e-008 3.2e-008 2.0e-009 1.00e+000  -7.668000027e+003 -7.668000015e+003 3.2e-008 0.01
6   1.3e-012 3.2e-012 2.0e-013 1.00e+000  -7.667999994e+003 -7.667999994e+003 3.2e-012 0.01
```

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

Using the same notation as in Sec. 13.2.2 the columns of the iteration log have the following meaning:

- `ITE`: Iteration index k.
- `PFEAS`: $\left\|Ax^k - b\tau^k\right\|_\infty$. The numbers in this column should converge monotonically towards zero but may stall at low level due to rounding errors.
- `DFEAS`: $\left\|A^T y^k + s^k - c\tau^k\right\|_\infty$. The numbers in this column should converge monotonically towards zero but may stall at low level due to rounding errors.
- `GFEAS`: $|-c^T x^k + b^T y^k - \kappa^k|$. The numbers in this column should converge monotonically towards zero but may stall at low level due to rounding errors.
- `PRSTATUS`: This number converges to 1 if the problem has an optimal solution whereas it converges to -1 if that is not the case.

- POBJ: $c^T x^k / \tau^k$. An estimate for the primal objective value.
- DOBJ: $b^T y^k / \tau^k$. An estimate for the dual objective value.
- MU: $\frac{(x^k)^T s^k + \tau^k \kappa^k}{n+1}$. The numbers in this column should always converge to zero.
- TIME: Time spent since the optimization started.

13.2.3 The Simplex Optimizer

An alternative to the interior-point optimizer is the simplex optimizer. The simplex optimizer uses a different method that allows exploiting an initial guess for the optimal solution to reduce the solution time. Depending on the problem it may be faster or slower to use an initial guess; see Sec. 13.2.1 for a discussion. **MOSEK** provides both a primal and a dual variant of the simplex optimizer.

Simplex Termination Criterion

The simplex optimizer terminates when it finds an optimal basic solution or an infeasibility certificate. A basic solution is optimal when it is primal and dual feasible; see Sec. 12.1 for a definition of the primal and dual problem. Due to the fact that computations are performed in finite precision **MOSEK** allows violations of primal and dual feasibility within certain tolerances. The user can control the allowed primal and dual tolerances with the parameters *MSK_DPAR_BASIS_TOL_X* and *MSK_DPAR_BASIS_TOL_S*.

Setting the parameter *MSK_IPAR_OPTIMIZER* to *"MSK_OPTIMIZER_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
  - *MSK_DPAR_BASIS_TOL_X*, and
  - *MSK_DPAR_BASIS_TOL_S*.
- Raise or lower pivot tolerance: Change the *MSK_DPAR_SIMPLEX_ABS_TOL_PIV* parameter.
- Switch optimizer: Try another optimizer.
- Switch off crash: Set both *MSK_IPAR_SIM_PRIMAL_CRASH* and *MSK_IPAR_SIM_DUAL_CRASH* to 0.
- Experiment with other pricing strategies: Try different values for the parameters

- *MSK_IPAR_SIM_PRIMAL_SELECTION* and
- *MSK_IPAR_SIM_DUAL_SELECTION*.

- If you are using warm-starts, in rare cases switching off this feature may improve stability. This is controlled by the *MSK_IPAR_SIM_HOTSTART* parameter.
- Increase maximum number of set-backs allowed controlled by *MSK_IPAR_SIM_MAX_NUM_SETBACKS*.
- If the problem repeatedly becomes infeasible try switching off the special degeneracy handling. See the parameter *MSK_IPAR_SIM_DEGEN* for details.

The Simplex Log

Below is a typical log output from the simplex optimizer:

```
Optimizer  - solved problem         : the primal
Optimizer  - Constraints            : 667
Optimizer  - Scalar variables       : 1424            conic                  : 0
Optimizer  - hotstart               : no
ITER      DEGITER(%)  PFEAS       DFEAS       POBJ                  DOBJ                  TIME␣
↪      TOTTIME
0         0.00        1.43e+05    NA          6.5584140832e+03      NA                    0.00␣
↪      0.02
1000      1.10        0.00e+00    NA          1.4588289726e+04      NA                    0.13␣
↪      0.14
2000      0.75        0.00e+00    NA          7.3705564855e+03      NA                    0.21␣
↪      0.22
3000      0.67        0.00e+00    NA          6.0509727712e+03      NA                    0.29␣
↪      0.31
4000      0.52        0.00e+00    NA          5.5771203906e+03      NA                    0.38␣
↪      0.39
4533      0.49        0.00e+00    NA          5.5018458883e+03      NA                    0.42␣
↪      0.44
```

The first lines summarize the problem the optimizer is solving. This is followed by the iteration log, with the following meaning:

- `ITER`: Number of iterations.
- `DEGITER(%)`: Ratio of degenerate iterations.
- `PFEAS`: Primal feasibility measure reported by the simplex optimizer. The numbers should be 0 if the problem is primal feasible (when the primal variant is used).
- `DFEAS`: Dual feasibility measure reported by the simplex optimizer. The number should be 0 if the problem is dual feasible (when the dual variant is used).
- `POBJ`: An estimate for the primal objective value (when the primal variant is used).
- `DOBJ`: An estimate for the dual objective value (when the dual variant is used).
- `TIME`: Time spent since this instance of the simplex optimizer was invoked (in seconds).
- `TOTTIME`: Time spent since optimization started (in seconds).

13.3 Conic Optimization - Interior-point optimizer

For conic optimization problems only an interior-point type optimizer is available.

13.3.1 The homogeneous primal-dual problem

The interior-point optimizer is an implementation of the so-called homogeneous and self-dual algorithm. For a detailed description of the algorithm, please see *[ART03]*. In order to keep our discussion simple we will assume that **MOSEK** solves a conic optimization problem of the form:

$$\begin{array}{lccc} \text{minimize} & c^T x & & \\ \text{subject to} & Ax & = & b, \\ & x \in \mathcal{K} & & \end{array} \tag{13.6}$$

where K is a convex cone. The corresponding dual problem is

$$\begin{array}{lccc} \text{maximize} & b^T y & & \\ \text{subject to} & A^T y + s & = & c, \\ & s \in \mathcal{K}^* & & \end{array} \tag{13.7}$$

where $\mathcal{K}^*$ is the dual cone of $\mathcal{K}$. See Sec. 12.2 for definitions.

Since it is not known beforehand whether problem (13.6) has an optimal solution, is primal infeasible or is dual infeasible, the optimization algorithm must deal with all three situations. This is the reason that **MOSEK** solves the so-called homogeneous model

$$\begin{array}{rcl} Ax - b\tau & = & 0, \\ A^T y + s - c\tau & = & 0, \\ -c^T x + b^T y - \kappa & = & 0, \\ x & \in & \mathcal{K}, \\ s & \in & \mathcal{K}^*, \\ \tau, \kappa & \geq & 0, \end{array} \tag{13.8}$$

where y and s correspond to the dual variables in (13.6), and τ and κ are two additional scalar variables. Note that the homogeneous model (13.8) always has a solution since

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

is a solution, although not a very interesting one. Any solution

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

to the homogeneous model (13.8) satisfies

$$(x^*)^T s^* + \tau^* \kappa^* = 0$$

i.e. complementarity. Observe that $x^* \in \mathcal{K}$ and $s^* \in \mathcal{K}^*$ implies

$$(x^*)^T s^* \geq 0$$

and therefore

$$\tau^* \kappa^* = 0.$$

since $\tau^*, \kappa^* \geq 0$. Hence, at least one of τ^* and κ^* is zero.

First, assume that $\tau^* > 0$ and hence $\kappa^* = 0$. It follows that

$$\begin{array}{rcl} A\frac{x^*}{\tau^*} & = & b, \\ A^T \frac{y^*}{\tau^*} + \frac{s^*}{\tau^*} & = & c, \\ -c^T \frac{x^*}{\tau^*} + b^T \frac{y^*}{\tau^*} & = & 0, \\ x^*/\tau^* & \in & \mathcal{K}, \\ s^*/\tau^* & \in & \mathcal{K}^*. \end{array}$$

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

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

is a primal-dual optimal solution.

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

$$\begin{array}{rcl} Ax^* & = & 0, \\ A^T y^* + s^* & = & 0, \\ -c^T x^* + b^T y^* & = & \kappa^*, \\ x^* & \in & \mathcal{K}, \\ s^* & \in & \mathcal{K}^*. \end{array}$$

This implies that at least one of

$$c^T x^* < 0 \tag{13.9}$$

or

$$b^T y^* > 0 \tag{13.10}$$

holds. If (13.9) is satisfied, then x^* is a certificate of dual infeasibility, whereas if (13.10) holds then y^* is a certificate of primal infeasibility.

In summary, by computing an appropriate solution to the homogeneous model, all information required for a solution to the original problem is obtained. A solution to the homogeneous model can be computed using a primal-dual interior-point algorithm *[And09]*.

13.3.2 Interior-point Termination Criterion

Since computations are performed in finite precision, and for efficiency reasons, it is not possible to solve the homogeneous model exactly in general. Hence, an exact optimal solution or an exact infeasibility certificate cannot be computed and a reasonable termination criterion has to be employed.

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

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

to the homogeneous model is generated, where

$$x^k \in \mathcal{K}, s^k \in \mathcal{K}^*, \tau^k, \kappa^k > 0.$$

Therefore, it is possible to compute the values:

$$\begin{array}{rcl} \rho_p^k & = & \arg\min_\rho \left\{ \rho \mid \left\| A \frac{x^k}{\tau^k} - b \right\|_\infty \le \rho \varepsilon_p (1 + \|b\|_\infty) \right\}, \\ \rho_d^k & = & \arg\min_\rho \left\{ \rho \mid \left\| A^T \frac{y^k}{\tau^k} + \frac{s^k}{\tau^k} - c \right\|_\infty \le \rho \varepsilon_d (1 + \|c\|_\infty) \right\}, \\ \rho_g^k & = & \arg\min_\rho \left\{ \rho \mid \left(\frac{(x^k)^T s^k}{(\tau^k)^2}, \left| \frac{c^T x^k}{\tau^k} - \frac{b^T y^k}{\tau^k} \right| \right) \le \rho \varepsilon_g \max \left(1, \frac{\min\left(\left|c^T x^k\right|, \left|b^T y^k\right| \right)}{\tau^k} \right) \right\}, \\ \rho_{pi}^k & = & \arg\min_\rho \left\{ \rho \mid \left\| A^T y^k + s^k \right\|_\infty \le \rho \varepsilon_i b^T y^k,\ b^T y^k > 0 \right\} \text{ and} \\ \rho_{di}^k & = & \arg\min_\rho \left\{ \rho \mid \left\| A x^k \right\|_\infty \le -\rho \varepsilon_i c^T x^k,\ c^T x^k < 0 \right\}. \end{array}$$

Note $\varepsilon_p, \varepsilon_d, \varepsilon_g$ and ε_i are nonnegative user specified tolerances.

Optimal Case

Observe ρ_p^k measures how far x^k/τ^k is from being a good approximate primal feasible solution. Indeed if $\rho_p^k \le 1$, then

$$\left\| A \frac{x^k}{\tau^k} - b \right\|_\infty \le \varepsilon_p (1 + \|b\|_\infty). \tag{13.11}$$

This shows the violations in the primal equality constraints for the solution x^k/τ^k is small compared to the size of b given ε_p is small.

Similarly, if $\rho_d^k \le 1$, then $(y^k, s^k)/\tau^k$ is an approximate dual feasible solution. If in addition $\rho_g \le 1$, then the solution $(x^k, y^k, s^k)/\tau^k$ is approximate optimal because the associated primal and dual objective values are almost identical.

In other words if $\max(\rho_p^k, \rho_d^k, \rho_g^k) \leq 1$, then

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

is an approximate optimal solution.

Dual Infeasibility Certificate

Next assume that $\rho_{di}^k \leq 1$ and hence

$$\left\|Ax^k\right\|_\infty \leq -\varepsilon_i c^T x^k \text{ and } -c^T x^k > 0$$

holds. Now in this case the problem is declared dual infeasible and x^k is reported as a certificate of dual infeasibility. The motivation for this stopping criterion is as follows. Let

$$\bar{x} := \frac{x^k}{-c^T x^k}$$

and it is easy to verify that

$$\|A\bar{x}\|_\infty \leq \varepsilon_i \text{ and } c^T\bar{x} = -1$$

which shows $\bar{x}$ is an approximate certificate of dual infeasibility, where ε_i controls the quality of the approximation.

Primal Infeasiblity Certificate

Next assume that $\rho_{pi}^k \leq 1$ and hence

$$\left\|A^T y^k + s^k\right\|_\infty \leq \varepsilon_i b^T y^k \text{ and } b^T y^k > 0$$

holds. Now in this case the problem is declared primal infeasible and (y^k, s^k) is reported as a certificate of primal infeasibility. The motivation for this stopping criterion is as follows. Let

$$\bar{y} := \frac{y^k}{b^T y^k} \text{ and } \bar{s} := \frac{s^k}{b^T y^k}$$

and it is easy to verify that

$$\left\|A^T\bar{y} + \bar{s}\right\|_\infty \leq \varepsilon_i \text{ and } b^T\bar{y} = 1$$

which shows (y^k, s^k) is an approximate certificate of dual infeasibility, where ε_i controls the quality of the approximation.

13.3.3 Adjusting optimality criteria

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

Table 13.2: Parameters employed in termination criterion

ToleranceParameter	name
ε_p	*MSK_DPAR_INTPNT_CO_TOL_PFEAS*
ε_d	*MSK_DPAR_INTPNT_CO_TOL_DFEAS*
ε_g	*MSK_DPAR_INTPNT_CO_TOL_REL_GAP*
ε_i	*MSK_DPAR_INTPNT_CO_TOL_INFEAS*

The default values of the termination tolerances are chosen such that for a majority of problems appearing in practice it is not possible to achieve much better accuracy. Therefore, tightening the tolerances usually is not worthwhile. However, an inspection of (13.11) reveals that the quality of the solution depends on $\|b\|_\infty$ and $\|c\|_\infty$; the smaller the norms are, the better the solution accuracy.

The interior-point method as implemented by **MOSEK** will converge toward optimality and primal and dual feasibility at the same rate *[And09]*. This means that if the optimizer is stopped prematurely then it is very unlikely that either the primal or dual solution is feasible. Another consequence is that in most cases all the tolerances, ε_p, ε_d, ε_g and ε_i, have to be relaxed together to achieve an effect.

If the optimizer terminates without locating a solution that satisfies the termination criteria, for example because of a stall or other numerical issues, then it will check if the solution found up to that point satisfies the same criteria with all tolerances multiplied by the value of *MSK_DPAR_INTPNT_CO_TOL_NEAR_REL* . If this is the case, the solution is still declared as optimal.

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

13.3.4 The Interior-point Log

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

```
Optimizer  - threads                : 20
Optimizer  - solved problem         : the primal
Optimizer  - Constraints            : 1
Optimizer  - Cones                  : 2
Optimizer  - Scalar variables       : 6                 conic                  : 6
Optimizer  - Semi-definite variables: 0                 scalarized             : 0
Factor     - setup time             : 0.00              dense det. time        : 0.00
Factor     - ML order time          : 0.00              GP order time          : 0.00
Factor     - nonzeros before factor : 1                 after factor           : 1
Factor     - dense dim.             : 0                 flops                  : 1.70e+01
ITE PFEAS    DFEAS    GFEAS    PRSTATUS   POBJ              DOBJ              MU       TIME
0   1.0e+00  2.9e-01  3.4e+00  0.00e+00   2.414213562e+00   0.000000000e+00   1.0e+00  0.01
1   2.7e-01  7.9e-02  2.2e+00  8.83e-01   6.969257574e-01   -9.685901771e-03  2.7e-01  0.01
2   6.5e-02  1.9e-02  1.2e+00  1.16e+00   7.606090061e-01   6.046141322e-01   6.5e-02  0.01
3   1.7e-03  5.0e-04  2.2e-01  1.12e+00   7.084385672e-01   7.045122560e-01   1.7e-03  0.01
4   1.4e-08  4.2e-09  4.9e-08  1.00e+00   7.071067941e-01   7.071067599e-01   1.4e-08  0.01
```

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

Using the same notation as in Sec. 13.3.1 the columns of the iteration log have the following meaning:

- `ITE`: Iteration index k.
- `PFEAS`: $\left\|Ax^k - b\tau^k\right\|_\infty$. The numbers in this column should converge monotonically towards zero but may stall at low level due to rounding errors.
- `DFEAS`: $\left\|A^Ty^k + s^k - c\tau^k\right\|_\infty$. The numbers in this column should converge monotonically towards zero but may stall at low level due to rounding errors.
- `GFEAS`: $|-c^Tx^k + b^Ty^k - \kappa^k|$. The numbers in this column should converge monotonically towards zero but may stall at low level due to rounding errors.
- `PRSTATUS`: This number converges to 1 if the problem has an optimal solution whereas it converges to -1 if that is not the case.
- `POBJ`: c^Tx^k/τ^k. An estimate for the primal objective value.
- `DOBJ`: b^Ty^k/τ^k. An estimate for the dual objective value.
- `MU`: $\frac{(x^k)^Ts^k+\tau^k\kappa^k}{n+1}$. The numbers in this column should always converge to zero.
- `TIME`: Time spent since the optimization started (in seconds).

13.4 The Optimizer for Mixed-integer Problems

A problem is a mixed-integer optimization problem when one or more of the variables are constrained to be integer valued. Readers unfamiliar with integer optimization are recommended to consult some relevant literature, e.g. the book *[Wol98]* by Wolsey.

13.4.1 The Mixed-integer Optimizer Overview

MOSEK can solve mixed-integer

- linear,
- quadratic and quadratically constrained, and
- conic

problems, except for mixed-integer semidefinite problems. The mixed-integer optimizer is specialized for solving linear and conic optimization problems. Pure quadratic and quadratically constrained problems are automatically converted to conic form.

By default the mixed-integer optimizer is run-to-run deterministic. This means that if a problem is solved twice on the same computer with identical parameter settings and no time limit then the obtained solutions will be identical. If a time limit is set then this may not be case since the time taken to solve a problem is not deterministic. The mixed-integer optimizer is parallelized i.e. it can exploit multiple cores during the optimization.

The solution process can be split into these phases:

1. **Presolve:** See Sec. 13.1.
2. **Cut generation:** Valid inequalities (cuts) are added to improve the lower bound.
3. **Heuristic:** Using heuristics the optimizer tries to guess a good feasible solution. Heuristics can be controlled by the parameter *MSK_IPAR_MIO_HEURISTIC_LEVEL* .
4. **Search:** The optimal solution is located by branching on integer variables.

13.4.2 Relaxations and bounds

It is important to understand that, in a worst-case scenario, the time required to solve integer optimization problems grows exponentially with the size of the problem (solving mixed-integer problems is NP-hard). For instance, a problem with n binary variables, may require time proportional to 2^n . The value of 2^n is huge even for moderate values of n.

In practice this implies that the focus should be on computing a near-optimal solution quickly rather than on locating an optimal solution. Even if the problem is only solved approximately, it is important to know how far the approximate solution is from an optimal one. In order to say something about the quality of an approximate solution the concept of *relaxation* is important.

Consider for example a mixed-integer optimization problem

$$\begin{array}{rccccl} z^* & = & \text{minimize} & c^T x & & \\ & & \text{subject to} & Ax & = & b, \\ & & & x \geq 0 & & \\ & & & x_j \in \mathbb{Z}, & & \forall j \in \mathcal{J}. \end{array} \tag{13.12}$$

It has the continuous relaxation

$$\begin{array}{rccccl} \underline{z} & = & \text{minimize} & c^T x & & \\ & & \text{subject to} & Ax & = & b, \\ & & & x \geq 0 & & \end{array} \tag{13.13}$$

obtained simply by ignoring the integrality restrictions. The relaxation is a continuous problem, and therefore much faster to solve to optimality with a linear (or, in the general case, conic) optimizer. We call the optimal value $\underline{z}$ the *objective bound*. The objective bound $\underline{z}$ normally increases during the solution search process when the continuous relaxation is gradually refined.

Moreover, if $\hat{x}$ is any feasible solution to (13.12) and

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

then

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

These two inequalities allow us to estimate the quality of the integer solution: it is no further away from the optimum than $\bar{z} - \underline{z}$ in terms of the objective value. Whenever a mixed-integer problem is solved **MOSEK** reports this lower bound so that the quality of the reported solution can be evaluated.

13.4.3 Outer approximation for mixed-integer conic problems

The relaxations of mixed integer conic problems can be solved either as a nonlinear problem with the interior point algorithm (default) or with a linear outer approximation algorithm. The type of relaxation used can be set with *MSK_IPAR_MIO_CONIC_OUTER_APPROXIMATION*. The best value for this option is highly problem dependent.

13.4.4 Randomization

A number of internal algorithms of the mixed-integer solver are dependend on random tie-breaking. The random tie-breaking can have a significant impact on the path taken by the algorithm and the optimal solution returned. The random seed can be set with the parameter *MSK_IPAR_MIO_SEED*.

13.4.5 Termination Criterion

In general, it is time consuming to find an exact feasible and optimal solution to an integer optimization problem, though in many practical cases it may be possible to find a sufficiently good solution. The issue of terminating the mixed-integer optimizer is rather delicate and the user has numerous possibilities of influencing it with various parameters. The mixed-integer optimizer employs a relaxed feasibility and optimality criterion to determine when a satisfactory solution is located.

A candidate solution that is feasible for the continuous relaxation is said to be an *integer feasible solution* if the criterion

$$\min(x_j - \lfloor x_j \rfloor, \lceil x_j \rceil - x_j) \leq \delta_1 \quad \forall j \in \mathcal{J}$$

is satisfied, meaning that x_j is at most δ_1 from the nearest integer.

Whenever the integer optimizer locates an integer feasible solution it will check if the criterion

$$\bar{z} - \underline{z} \leq \max(\delta_2, \delta_3 \max(\delta_4, |\bar{z}|))$$

is satisfied. If this is the case, the integer optimizer terminates and reports the integer feasible solution as an optimal solution.

All the δ tolerances discussed above can be adjusted using suitable parameters — see Table 13.3.

Table 13.3: Tolerances for the mixed-integer optimizer.

Tolerance	Parameter name
δ_1	*MSK_DPAR_MIO_TOL_ABS_RELAX_INT*
δ_2	*MSK_DPAR_MIO_TOL_ABS_GAP*
δ_3	*MSK_DPAR_MIO_TOL_REL_GAP*
δ_4	*MSK_DPAR_MIO_REL_GAP_CONST*

In Table 13.4 some other common parameters affecting the integer optimizer termination criterion are shown.

Table 13.4: Other parameters affecting the integer optimizer termination criterion.

Parameter name	Explanation
MSK_IPAR_MIO_MAX_NUM_BRANCHES	Maximum number of branches allowed.
MSK_IPAR_MIO_MAX_NUM_RELAXS	Maximum number of relaxations allowed.
MSK_IPAR_MIO_MAX_NUM_SOLUTIONS	Maximum number of feasible integer solutions allowed.

13.4.6 Speeding Up the Solution Process

As mentioned previously, in many cases it is not possible to find an optimal solution to an integer optimization problem in a reasonable amount of time. Some suggestions to reduce the solution time are:

- Relax the termination criterion: In case the run time is not acceptable, the first thing to do is to relax the termination criterion — see Sec. 13.4.5 for details.

- Specify a good initial solution: In many cases a good feasible solution is either known or easily computed using problem-specific knowledge. If a good feasible solution is known, it is usually worthwhile to use this as a starting point for the integer optimizer. See Sec. 6.9.2.
- Improve the formulation: A mixed-integer optimization problem may be impossible to solve in one form and quite easy in another form. However, it is beyond the scope of this manual to discuss good formulations for mixed-integer problems. For discussions on this topic see for example *[Wol98]*.

13.4.7 Understanding Solution Quality

To determine the quality of the solution one should check the following:

- The problem status and solution status returned by **MOSEK**, as well as constraint violations in case of suboptimal solutions.
- The *optimality gap* defined as

  $$\epsilon = |(\text{objective value of feasible solution}) - (\text{objective bound})| = |\bar{z} - \underline{z}|.$$

  which measures how much the located solution can deviate from the optimal solution to the problem. The optimality gap can be retrieved through the information item *"MSK_DINF_MIO_OBJ_ABS_GAP"*. Often it is more meaningful to look at the relative optimality gap normalized against the magnitude of the solution.

  $$\epsilon_{\text{rel}} = \frac{|\bar{z} - \underline{z}|}{\max(\delta_4, |\bar{z}|)}.$$

  The relative optimality gap is available in the information item *"MSK_DINF_MIO_OBJ_REL_GAP"*.

13.4.8 The Mixed-integer Log

Below is a typical log output from the mixed-integer optimizer:

```
Presolved problem: 6573 variables, 35728 constraints, 101258 non-zeros
Presolved problem: 0 general integer, 4294 binary, 2279 continuous
Clique table size: 1636
BRANCHES RELAXS   ACT_NDS  DEPTH    BEST_INT_OBJ          BEST_RELAX_OBJ        REL_GAP(%)  TIME
0        1        0        0        NA                    1.8218819866e+07      NA          1.6
0        1        0        0        1.8331557950e+07      1.8218819866e+07      0.61        3.5
0        1        0        0        1.8300507546e+07      1.8218819866e+07      0.45        4.3
Cut generation started.
0        2        0        0        1.8300507546e+07      1.8218819866e+07      0.45        5.3
Cut generation terminated. Time = 1.43
0        3        0        0        1.8286893047e+07      1.8231580587e+07      0.30        7.5
15       18       1        0        1.8286893047e+07      1.8231580587e+07      0.30        10.5
31       34       1        0        1.8286893047e+07      1.8231580587e+07      0.30        11.1
51       54       1        0        1.8286893047e+07      1.8231580587e+07      0.30        11.6
91       94       1        0        1.8286893047e+07      1.8231580587e+07      0.30        12.4
171      174      1        0        1.8286893047e+07      1.8231580587e+07      0.30        14.3
331      334      1        0        1.8286893047e+07      1.8231580587e+07      0.30        17.9

[ ... ]

Objective of best integer solution : 1.825846762609e+07
Best objective bound               : 1.823311032986e+07
Construct solution objective       : Not employed
Construct solution # roundings     : 0
User objective cut value           : 0
Number of cuts generated           : 117
  Number of Gomory cuts            : 108
  Number of CMIR cuts              : 9
```

(continues on next page)

(continued from previous page)

```
Number of branches                 : 4425
Number of relaxations solved       : 4410
Number of interior point iterations: 25
Number of simplex iterations       : 221131
```

The first lines contain a summary of the problem as seen by the optimizer. This is followed by the iteration log. The columns have the following meaning:

- `BRANCHES`: Number of branches generated.
- `RELAXS`: Number of relaxations solved.
- `ACT_NDS`: Number of active branch bound nodes.
- `DEPTH`: Depth of the recently solved node.
- `BEST_INT_OBJ`: The best integer objective value, $\bar{z}$.
- `BEST_RELAX_OBJ`: The best objective bound, $\underline{z}$.
- `REL_GAP(%)`: Relative optimality gap, $100\% \cdot \epsilon_{\mathrm{rel}}$
- `TIME`: Time (in seconds) from the start of optimization.

Following that a summary of the optimization process is printed.

Chapter 14

Additional features

In this section we describe additional features and tools which enable more detailed analysis of optimization problems with **MOSEK**.

14.1 Problem Analyzer

The problem analyzer prints a survey of the structure of the problem, with information about linear constraints and objective, quadratic constraints, conic constraints and variables.

In the initial stages of model formulation the problem analyzer may be used as a quick way of verifying that the model has been built or imported correctly. In later stages it can help revealing special structures within the model that may be used to tune the optimizer's performance or to identify the causes of numerical difficulties.

The problem analyzer is run using the *mosekopt* `('anapro')` command and produces output similar to the following (this is the problem analyzer's survey of the `aflow30a` problem from the MIPLIB 2003 collection).

```
                    Analyzing the problem

*** Structural report
               Dimensions
               Constraints    Variables     Matrix var.   Cones
               479            842           0             0

               Constraint and bound types
               Free           Lower         Upper         Ranged        Fixed
 Constraints:  0              0             421           0             58
   Variables:  0              0             0             842           0

               Integer constraint types
               Binary         General
               421            0

*** Data report
               Nonzeros       Min           Max
        |cj|:  421            1.1e+01       5.0e+02
       |Aij|:  2091           1.0e+00       1.0e+02

               # finite       Min           Max
      |blci|:  58             1.0e+00       1.0e+01
      |buci|:  479            0.0e+00       1.0e+01
      |blxj|:  842            0.0e+00       0.0e+00
      |buxj|:  842            1.0e+00       1.0e+02

*** Done analyzing the problem
```

The survey is divided into a structural and numerical report. The content should be self-explanatory.

14.2 Automatic Repair of Infeasible Problems

MOSEK provides an automatic repair tool for infeasible linear problems which we cover in this section. Note that most infeasible models are so due to bugs which can (and should) be more reliably fixed manually, using the knowledge of the model structure. We discuss this approach in Sec. 8.3.

14.2.1 Automatic repair

The main idea can be described as follows. Consider the linear optimization problem with m constraints and n variables

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

which is assumed to be infeasible.

One way of making the problem feasible is to reduce the lower bounds and increase the upper bounds. If the change is sufficiently large the problem becomes feasible. Now an obvious idea is to compute the optimal relaxation by solving an optimization problem. The problem

$$
\begin{array}{lcccl}
\text{minimize} & & p(v_l^c, v_u^c, v_l^x, v_u^x) & & \\
\text{subject to} & l^c \leq & Ax + v_l^c - v_u^c & \leq & u^c, \\
& l^x \leq & x + v_l^x - v_u^x & \leq & u^x, \\
& & v_l^c, v_u^c, v_l^x, v_u^x \geq 0 & &
\end{array}
\tag{14.1}
$$

does exactly that. The additional variables $(v_l^c)_i$, $(v_u^c)_i$, $(v_l^x)_j$ and $(v_u^c)_j$ are *elasticity* variables because they allow a constraint to be violated and hence add some elasticity to the problem. For instance, the elasticity variable $(v_l^c)_i$ controls how much the lower bound $(l^c)_i$ should be relaxed to make the problem feasible. Finally, the so-called penalty function

$$
p(v_l^c, v_u^c, v_l^x, v_u^x)
$$

is chosen so it penalizes changes to bounds. Given the weights

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

a natural choice is

$$
p(v_l^c, v_u^c, v_l^x, v_u^x) = (w_l^c)^T v_l^c + (w_u^c)^T v_u^c + (w_l^x)^T v_l^x + (w_u^x)^T v_u^x.
$$

Hence, the penalty function $p()$ is a weighted sum of the elasticity variables and therefore the problem (14.1) keeps the amount of relaxation at a minimum. Please observe that

- the problem (14.1) is always feasible.
- a negative weight implies problem (14.1) is unbounded. For this reason if the value of a weight is negative **MOSEK** fixes the associated elasticity variable to zero. Clearly, if one or more of the weights are negative, it may imply that it is not possible to repair the problem.

A simple choice of weights is to set them all to 1, but of course that does not take into account that constraints may have different importance.

Caveats

Observe if the infeasible problem

$$\begin{array}{lccc} \text{minimize} & x+z & & \\ \text{subject to} & x & = & -1, \\ & x & \geq & 0 \end{array}$$

is repaired then it will become unbounded. Hence, a repaired problem may not have an optimal solution.

Another and more important caveat is that only a minimal repair is performed i.e. the repair that barely makes the problem feasible. Hence, the repaired problem is barely feasible and that sometimes makes the repaired problem hard to solve.

Using the automatic repair tool

In this subsection we consider an infeasible linear optimization example:

$$\begin{array}{lccccc} \text{minimize} & -10x_1 & & -9x_2, & & \\ \text{subject to} & 7/10x_1 & + & 1x_2 & \leq & 630, \\ & 1/2x_1 & + & 5/6x_2 & \leq & 600, \\ & 1x_1 & + & 2/3x_2 & \leq & 708, \\ & 1/10x_1 & + & 1/4x_2 & \leq & 135, \\ & x_1, & & x_2 & \geq & 0, \\ & & & x_2 & \geq & 650. \end{array} \tag{14.2}$$

The code following code will form the repaired problem and solve it.

Listing 14.1: An example of feasibility repair applied to problem (14.2).

```
function feasrepairex1(inputfile)

cmd = sprintf('read(%s)', inputfile);
[r,res]=mosekopt(cmd);

res.prob.primalrepair = [];
res.prob.primalrepair.wux = [1,1];
res.prob.primalrepair.wlx = [1,1];
res.prob.primalrepair.wuc = [1,1,1,1];
res.prob.primalrepair.wlc = [1,1,1,1];

param.MSK_IPAR_LOG_FEAS_REPAIR = 3;
[r,res]=mosekopt('minimize primalrepair',res.prob,param);
fprintf('Return code: %d\n',r);

end
```

The parameter *MSK_IPAR_LOG_FEAS_REPAIR* controls the amount of log output from the repair. A value of 2 causes the optimal repair to printed out. If the fields `wlx`, `wux`, `wlc` or `wuc` are not specified, they are all assumed to be 1-vectors of appropriate dimensions.

The above code will produce the following log report:

```
MOSEK Version 9.0.0.25(ALPHA) (Build date: 2017-11-7 16:11:50)
Copyright (c) MOSEK ApS, Denmark. WWW: mosek.com
Platform: Linux/64-X86

Open file 'feasrepair.lp'
Reading started.
Reading terminated. Time: 0.00

Read summary
  Type              : LO (linear optimization problem)
```

(continues on next page)

(continued from previous page)

```
  Objective sense  : min
  Scalar variables : 2
  Matrix variables : 0
  Constraints      : 4
  Cones            : 0
  Time             : 0.0

Problem
  Name                   :
  Objective sense        : min
  Type                   : LO (linear optimization problem)
  Constraints            : 4
  Cones                  : 0
  Scalar variables       : 2
  Matrix variables       : 0
  Integer variables      : 0

Primal feasibility repair started.
Optimizer started.
Presolve started.
Linear dependency checker started.
Linear dependency checker terminated.
Eliminator started.
Freed constraints in eliminator : 2
Eliminator terminated.
Eliminator - tries                  : 1                 time                   : 0.00
Lin. dep.  - tries                  : 1                 time                   : 0.00
Lin. dep.  - number                 : 0
Presolve terminated. Time: 0.00
Problem
  Name                   :
  Objective sense        : min
  Type                   : LO (linear optimization problem)
  Constraints            : 8
  Cones                  : 0
  Scalar variables       : 14
  Matrix variables       : 0
  Integer variables      : 0

Optimizer  - threads                : 20
Optimizer  - solved problem         : the primal
Optimizer  - Constraints            : 2
Optimizer  - Cones                  : 0
Optimizer  - Scalar variables       : 5                 conic                  : 0
Optimizer  - Semi-definite variables: 0                 scalarized             : 0
Factor     - setup time             : 0.00              dense det. time        : 0.00
Factor     - ML order time          : 0.00              GP order time          : 0.00
Factor     - nonzeros before factor : 3                 after factor           : 3
Factor     - dense dim.             : 0                 flops                  : 5.00e+01
ITE PFEAS    DFEAS    GFEAS    PRSTATUS   POBJ              DOBJ              MU       TIME
0   2.7e+01  1.0e+00  4.0e+00  1.00e+00   3.000000000e+00   0.000000000e+00   1.0e+00  0.00
1   2.5e+01  9.1e-01  1.4e+00  0.00e+00   8.711262850e+00   1.115287830e+01   2.4e+00  0.00
2   2.4e+00  8.8e-02  1.4e-01  -7.33e-01  4.062505701e+01   4.422203730e+01   2.3e-01  0.00
3   9.4e-02  3.4e-03  5.5e-03  1.33e+00   4.250700434e+01   4.258548510e+01   9.1e-03  0.00
4   2.0e-05  7.2e-07  1.1e-06  1.02e+00   4.249996599e+01   4.249998669e+01   1.9e-06  0.00
5   2.0e-09  7.2e-11  1.1e-10  1.00e+00   4.250000000e+01   4.250000000e+01   1.9e-10  0.00
Basis identification started.
Basis identification terminated. Time: 0.00
Optimizer terminated. Time: 0.01

Basic solution summary
```

(continues on next page)

(continued from previous page)

```
  Problem status  : PRIMAL_AND_DUAL_FEASIBLE
  Solution status : OPTIMAL
  Primal.  obj: 4.2500000000e+01    nrm: 6e+02    Viol.  con: 1e-13    var: 0e+00
  Dual.    obj: 4.2499999999e+01    nrm: 2e+00    Viol.  con: 0e+00    var: 9e-11
Optimal objective value of the penalty problem: 4.250000000000e+01

Repairing bounds.
Increasing the upper bound 1.35e+02 on constraint 'c4' (3) with 2.25e+01.
Decreasing the lower bound 6.50e+02 on variable 'x2' (4) with 2.00e+01.
Primal feasibility repair terminated.
Optimizer started.
Optimizer terminated. Time: 0.00


Interior-point solution summary
  Problem status  : PRIMAL_AND_DUAL_FEASIBLE
  Solution status : OPTIMAL
  Primal.  obj: -5.6700000000e+03   nrm: 6e+02    Viol.  con: 0e+00    var: 0e+00
  Dual.    obj: -5.6700000000e+03   nrm: 1e+01    Viol.  con: 0e+00    var: 0e+00

Basic solution summary
  Problem status  : PRIMAL_AND_DUAL_FEASIBLE
  Solution status : OPTIMAL
  Primal.  obj: -5.6700000000e+03   nrm: 6e+02    Viol.  con: 0e+00    var: 0e+00
  Dual.    obj: -5.6700000000e+03   nrm: 1e+01    Viol.  con: 0e+00    var: 0e+00

Optimizer summary
  Optimizer                 -                        time: 0.00
    Interior-point          - iterations : 0         time: 0.00
      Basis identification  -                        time: 0.00
        Primal              - iterations : 0         time: 0.00
        Dual                - iterations : 0         time: 0.00
        Clean primal        - iterations : 0         time: 0.00
        Clean dual          - iterations : 0         time: 0.00
    Simplex                 -                        time: 0.00
      Primal simplex        - iterations : 0         time: 0.00
      Dual simplex          - iterations : 0         time: 0.00
    Mixed integer           - relaxations: 0         time: 0.00
```

It will also modify the task according to the optimal elasticity variables found. In this case the optimal repair it is to increase the upper bound on constraint `c4` by 22.5 and decrease the lower bound on variable `x2` by 20.

14.3 Sensitivity Analysis

Given an optimization problem it is often useful to obtain information about how the optimal objective value changes when the problem parameters are perturbed. E.g, assume that a bound represents the capacity of a machine. Now, it may be possible to expand the capacity for a certain cost and hence it is worthwhile knowing what the value of additional capacity is. This is precisely the type of questions the sensitivity analysis deals with.

Analyzing how the optimal objective value changes when the problem data is changed is called *sensitivity analysis*.

References

The book *[Chv83]* discusses the classical sensitivity analysis in Chapter 10 whereas the book *[RTV97]* presents a modern introduction to sensitivity analysis. Finally, it is recommended to read the short paper *[Wal00]* to avoid some of the pitfalls associated with sensitivity analysis.

Warning: Currently, sensitivity analysis is only available for continuous linear optimization problems. Moreover, **MOSEK** can only deal with perturbations of bounds and objective function coefficients.

14.3.1 Sensitivity Analysis for Linear Problems

The Optimal Objective Value Function

Assume that we are given the problem

$$\begin{array}{rclccccc} z(l^c, u^c, l^x, u^x, c) & = & \text{minimize} & & & c^T x & & \\ & & \text{subject to} & l^c & \leq & Ax & \leq & u^c, \\ & & & l^x & \leq & x & \leq & u^x, \end{array} \tag{14.3}$$

and we want to know how the optimal objective value changes as l_i^c is perturbed. To answer this question we define the perturbed problem for l_i^c as follows

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

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

$$f_{l_i^c}(\beta) \tag{14.4}$$

shows the optimal objective value as a function of β. Please note that a change in β corresponds to a perturbation in l_i^c and hence (14.4) shows the optimal objective value as a function of varying l_i^c with the other bounds fixed.

It is possible to prove that the function (14.4) is a piecewise linear and convex function, i.e. its graph may look like in Fig. 14.1 and Fig. 14.2.

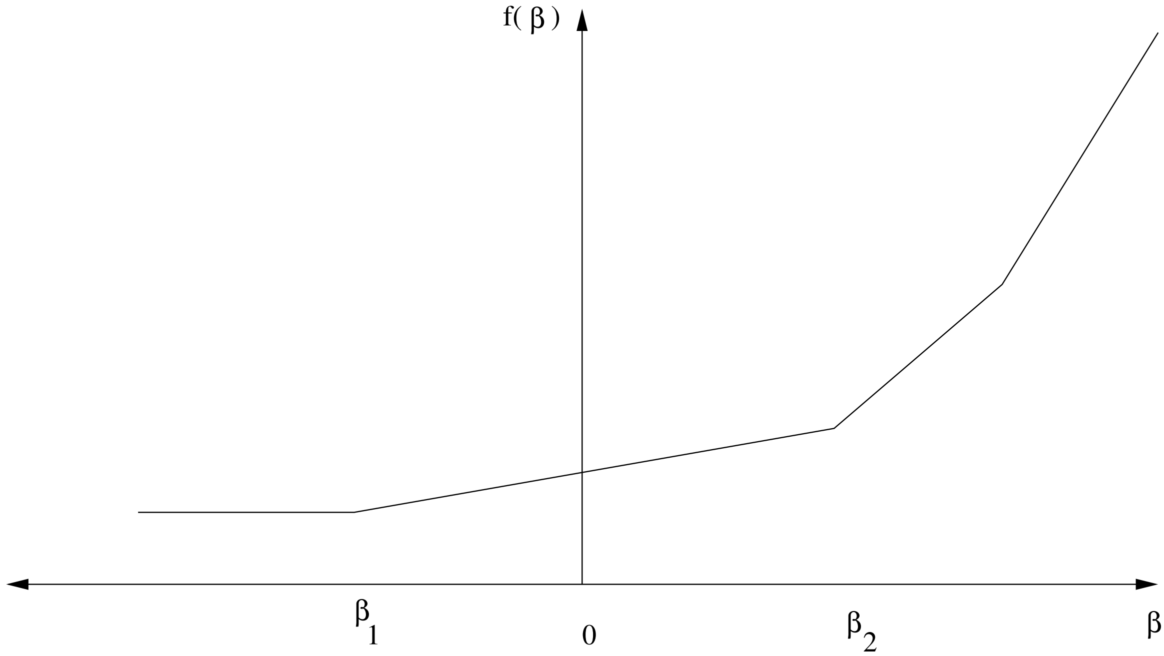

Fig. 14.1: $\beta = 0$ is in the interior of linearity interval.

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

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

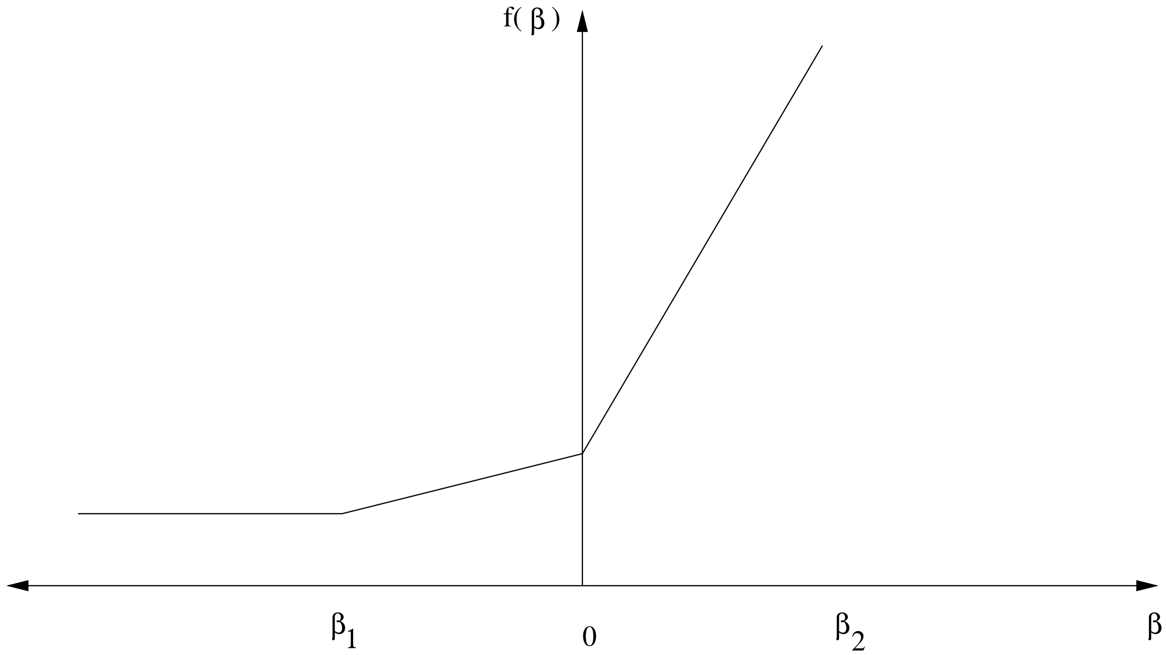

Fig. 14.2: $\beta = 0$ is a breakpoint.

which is called the *shadow price* related to l_i^c. The shadow price specifies how the objective value changes for small changes of β around zero. Moreover, we are interested in the *linearity interval*

$$\beta \in [\beta_1, \beta_2]$$

for which

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

Since $f_{l_i^c}$ is not a smooth function $f'_{l_i^c}$ may not be defined at 0, as illustrated in Fig. 14.2. In this case we can define a left and a right shadow price and a left and a right linearity interval.

The function $f_{l_i^c}$ considered only changes in l_i^c. We can define similar functions for the remaining parameters of the z defined in (14.3) as well:

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

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

Equality Constraints

In **MOSEK** a constraint can be specified as either an equality constraint or a ranged constraint. If some constraint e_i^c is an equality constraint, we define the optimal value function for this constraint as

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

Thus for an equality constraint the upper and the lower bounds (which are equal) are perturbed simultaneously. Therefore, **MOSEK** will handle sensitivity analysis differently for a ranged constraint with $l_i^c = u_i^c$ and for an equality constraint.

The Basis Type Sensitivity Analysis

The classical sensitivity analysis discussed in most textbooks about linear optimization, e.g. *[Chv83]*, is based on an optimal basis. This method may produce misleading results *[RTV97]* but is computationally cheap. This is the type of sensitivity analysis implemented in **MOSEK**.

We will now briefly discuss the basis type sensitivity analysis. Given an optimal basic solution which provides a partition of variables into basic and non-basic variables, the basis type sensitivity analysis computes the linearity interval $[\beta_1, \beta_2]$ so that the basis remains optimal for the perturbed problem. A shadow price associated with the linearity interval is also computed. However, it is well-known that an optimal basic solution may not be unique and therefore the result depends on the optimal basic solution employed in the sensitivity analysis. If the optimal objective value function has a breakpoint for $\beta = 0$ then the basis type sensitivity method will only provide a subset of either the left or the right linearity interval.

In summary, the basis type sensitivity analysis is computationally cheap but does not provide complete information. Hence, the results of the basis type sensitivity analysis should be used with care.

Example: Sensitivity Analysis

As an example we will use the following transportation problem. Consider the problem of minimizing the transportation cost between a number of production plants and stores. Each plant supplies a number of goods and each store has a given demand that must be met. Supply, demand and cost of transportation per unit are shown in Fig. 14.3.

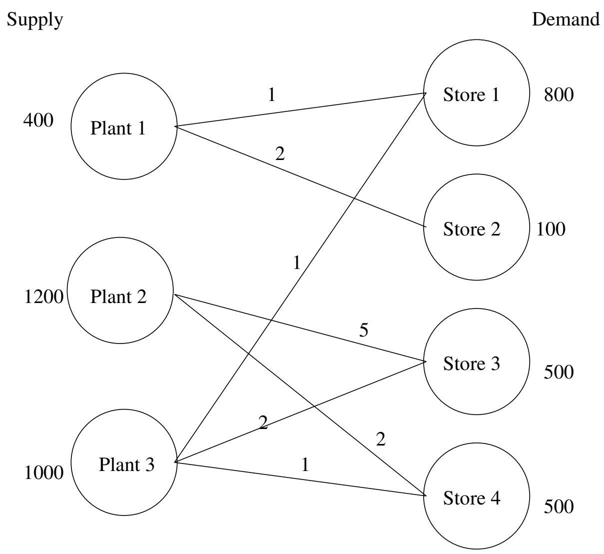

Fig. 14.3: Supply, demand and cost of transportation.

If we denote the number of transported goods from location i to location j by x_{ij}, problem can be formulated as the linear optimization problem of minimizing

$$1x_{11} + 2x_{12} + 5x_{23} + 2x_{24} + 1x_{31} + 2x_{33} + 1x_{34}$$

subject to

$$\begin{array}{ccccccccccccccc}
x_{11} & + & x_{12} & & & & & & & & & & & \leq & 400, \\
 & & & & x_{23} & + & x_{24} & & & & & & & \leq & 1200, \\
 & & & & & & & & x_{31} & + & x_{33} & + & x_{34} & \leq & 1000, \\
x_{11} & & & & & & & + & x_{31} & & & & & = & 800, \\
 & & x_{12} & & & & & & & & & & & = & 100, \\
 & & & & x_{23} & + & & & & & x_{33} & & & = & 500, \\
 & & & & & & x_{24} & + & & & & & x_{34} & = & 500, \\
x_{11}, & & x_{12}, & & x_{23}, & & x_{24}, & & x_{31}, & & x_{33}, & & x_{34} & \geq & 0.
\end{array} \tag{14.5}$$

The sensitivity parameters are shown in Table 14.1 and Table 14.2.

Table 14.1: Ranges and shadow prices related to bounds on constraints and variables.

Con.	β_1	β_2	σ_1	σ_2
1	-300.00	0.00	3.00	3.00
2	-700.00	$+\infty$	0.00	0.00
3	-500.00	0.00	3.00	3.00
4	-0.00	500.00	4.00	4.00
5	-0.00	300.00	5.00	5.00
6	-0.00	700.00	5.00	5.00
7	-500.00	700.00	2.00	2.00
Var.	β_1	β_2	σ_1	σ_2
x_{11}	$-\infty$	300.00	0.00	0.00
x_{12}	$-\infty$	100.00	0.00	0.00
x_{23}	$-\infty$	0.00	0.00	0.00
x_{24}	$-\infty$	500.00	0.00	0.00
x_{31}	$-\infty$	500.00	0.00	0.00
x_{33}	$-\infty$	500.00	0.00	0.00
x_{34}	-0.000000	500.00	2.00	2.00

Table 14.2: Ranges and shadow prices related to the objective coefficients.

Var.	β_1	β_2	σ_1	σ_2
c_1	$-\infty$	3.00	300.00	300.00
c_2	$-\infty$	∞	100.00	100.00
c_3	-2.00	∞	0.00	0.00
c_4	$-\infty$	2.00	500.00	500.00
c_5	-3.00	∞	500.00	500.00
c_6	$-\infty$	2.00	500.00	500.00
c_7	-2.00	∞	0.00	0.00

Examining the results from the sensitivity analysis we see that for constraint number 1 we have $\sigma_1 = 3$ and $\beta_1 = -300,\ \beta_2 = 0$.

If the upper bound on constraint 1 is decreased by

$$\beta \in [0, 300]$$

then the optimal objective value will increase by the value

$$\sigma_1 \beta = 3\beta.$$

14.3.2 Sensitivity Analysis with MOSEK

The following describe sensitivity analysis from the MATLAB toolbox.

On bounds

The index of bounds/variables to analyzed for sensitivity are specified in the following subfields of the MATLAB structure `prob`:

- `.prisen.cons.subu` Indexes of constraints, where upper bounds are analyzed for sensitivity.
- `.prisen.cons.subl` Indexes of constraints, where lower bounds are analyzed for sensitivity.

- `.prisen.vars.subu` Indexes of variables, where upper bounds are analyzed for sensitivity.
- `.prisen.vars.subl` Indexes of variables, where lower bounds are analyzed for sensitivity.
- `.duasen.sub` Index of variables where coefficients are analyzed for sensitivity.

For an equality constraint, the index can be specified in either `subu` or `subl`. After calling *mosekopt* the results are returned in the subfields `prisen` and `duasen` of `res`.

prisen

The field `prisen` is structured as follows:

- `.cons`: a MATLAB structure with subfields:
  - `.lr_bl` Left value β_1 in the linearity interval for a lower bound.
  - `.rr_bl` Right value β_2 in the linearity interval for a lower bound.
  - `.ls_bl` Left shadow price s_l for a lower bound.
  - `.rs_bl` Right shadow price s_r for a lower bound.
  - `.lr_bu` Left value β_1 in the linearity interval for an upper bound.
  - `.rr_bu` Right value β_2 in the linearity interval for an upper bound.
  - `.ls_bu` Left shadow price s_l for an upper bound.
  - `.rs_bu` Right shadow price s_r for an upper bound.
- `.var`: MATLAB structure with subfields:
  - `.lr_bl` Left value β_1 in the linearity interval for a lower bound on a varable.
  - `.rr_bl` Right value β_2 in the linearity interval for a lower bound on a varable.
  - `.ls_bl` Left shadow price s_l for a lower bound on a varable.
  - `.rs_bl` Right shadow price s_r for lower bound on a varable.
  - `.lr_bu` Left value β_1 in the linearity interval for an upper bound on a varable.
  - `.rr_bu` Right value β_2 in the linearity interval for an upper bound on a varable.
  - `.ls_bu` Left shadow price s_l for an upper bound on a varables.
  - `.rs_bu` Right shadow price s_r for an upper bound on a varables.

duasen

The field `duasen` is structured as follows:

- `.lr_c` Left value β_1 of linearity interval for an objective coefficient.
- `.rr_c` Right value β_2 of linearity interval for an objective coefficient.
- `.ls_c` Left shadow price s_l for an objective coefficients .
- `.rs_c` Right shadow price s_r for an objective coefficients.

Example

Consider the problem defined in (14.5). Suppose we wish to perform sensitivity analysis on all bounds and coefficients. The following example demonstrates this as well as the method for changing between basic and full sensitivity analysis.

Listing 14.2: A script to perform sensitivity analysis on problem (14.5).

```
function sensitivity()

clear prob;

% Obtain all symbolic constants
% defined by MOSEK.
[r,res]  = mosekopt('symbcon');
sc       = res.symbcon;

prob.blc = [-Inf, -Inf, -Inf, 800,100,500,500];
prob.buc = [ 400, 1200, 1000, 800,100,500,500];
prob.c   = [1.0,2.0,5.0,2.0,1.0,2.0,1.0]';
prob.blx = [0.0,0.0,0.0,0.0,0.0,0.0,0.0];
prob.bux = [Inf,Inf,Inf,Inf, Inf,Inf,Inf];

subi     = [  1,  1,  2,  2,  3,  3,  3,  4,  4,  5,  6,  6,  7,  7];
subj     = [  1,  2,  3,  4,  5,  6,  7,  1,  5,  6,  3,  6,  4,  7];
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];

prob.a = sparse(subi,subj,val);

% analyse upper bound 1:7
prob.prisen.cons.subl = [];
prob.prisen.cons.subu = [1:7];
% analyse lower bound on variables 1:7
prob.prisen.vars.subl = [1:7];
prob.prisen.vars.subu = [];
% analyse coeficient 1:7
prob.duasen.sub = [1:7];
[r,res] = mosekopt('minimize echo(0)',prob);

%Print results

fprintf('\nBasis sensitivity results:\n')
fprintf('\nSensitivity for bounds on constraints:\n')
for i = 1:length(prob.prisen.cons.subl)
  fprintf (...
  'con = %d, beta_1 = %.1f, beta_2 = %.1f, delta_1 = %.1f,delta_2 = %.1f\n', ...
  prob.prisen.cons.subl(i),res.prisen.cons.lr_bl(i), ...
  res.prisen.cons.rr_bl(i),...
  res.prisen.cons.ls_bl(i),...
  res.prisen.cons.rs_bl(i));
end

for i = 1:length(prob.prisen.cons.subu)
  fprintf (...
  'con = %d, beta_1 = %.1f, beta_2 = %.1f, delta_1 = %.1f,delta_2 = %.1f\n', ...
  prob.prisen.cons.subu(i),res.prisen.cons.lr_bu(i), ...
  res.prisen.cons.rr_bu(i),...
  res.prisen.cons.ls_bu(i),...
  res.prisen.cons.rs_bu(i));
end
fprintf('Sensitivity for bounds on variables:\n')
for i = 1:length(prob.prisen.vars.subl)
fprintf (...
'var = %d, beta_1 = %.1f, beta_2 = %.1f, delta_1 = %.1f,delta_2 = %.1f\n', ...
 prob.prisen.vars.subl(i),res.prisen.vars.lr_bl(i), ...
 res.prisen.vars.rr_bl(i),...
 res.prisen.vars.ls_bl(i),...
```

(continues on next page)

(continued from previous page)

```
 res.prisen.vars.rs_bl(i));
end

for i = 1:length(prob.prisen.vars.subu)
  fprintf (...
  'var = %d, beta_1 = %.1f, beta_2 = %.1f, delta_1 = %.1f,delta_2 = %.1f\n', ...
  prob.prisen.vars.subu(i),res.prisen.vars.lr_bu(i), ...
  res.prisen.vars.rr_bu(i),...
  res.prisen.vars.ls_bu(i),...
  res.prisen.vars.rs_bu(i));
end

fprintf('Sensitivity for coefficients in objective:\n')
for i = 1:length(prob.duasen.sub)
  fprintf (...
  'var = %d, beta_1 = %.1f, beta_2 = %.1f, delta_1 = %.1f,delta_2 = %.1f\n', ...
  prob.duasen.sub(i),res.duasen.lr_c(i), ...
  res.duasen.rr_c(i),...
  res.duasen.ls_c(i),...
  res.duasen.rs_c(i));
end
```

The output from running the example in Listing 14.2 is shown below.

```
Sensitivity for bounds on constraints:
con = 1, beta_1 = -300.0, beta_2 = 0.0, delta_1 = 3.0,delta_2 = 3.0
con = 2, beta_1 = -700.0, beta_2 = Inf, delta_1 = 0.0,delta_2 = 0.0
con = 3, beta_1 = -500.0, beta_2 = 0.0, delta_1 = 3.0,delta_2 = 3.0
con = 4, beta_1 = -0.0, beta_2 = 500.0, delta_1 = 4.0,delta_2 = 4.0
con = 5, beta_1 = -0.0, beta_2 = 300.0, delta_1 = 5.0,delta_2 = 5.0
con = 6, beta_1 = -0.0, beta_2 = 700.0, delta_1 = 5.0,delta_2 = 5.0
con = 7, beta_1 = -500.0, beta_2 = 700.0, delta_1 = 2.0,delta_2 = 2.0
Sensitivity for bounds on variables:
var = 1, beta_1 = Inf, beta_2 = 300.0, delta_1 = 0.0,delta_2 = 0.0
var = 2, beta_1 = Inf, beta_2 = 100.0, delta_1 = 0.0,delta_2 = 0.0
var = 3, beta_1 = Inf, beta_2 = 0.0, delta_1 = 0.0,delta_2 = 0.0
var = 4, beta_1 = Inf, beta_2 = 500.0, delta_1 = 0.0,delta_2 = 0.0
var = 5, beta_1 = Inf, beta_2 = 500.0, delta_1 = 0.0,delta_2 = 0.0
var = 6, beta_1 = Inf, beta_2 = 500.0, delta_1 = 0.0,delta_2 = 0.0
var = 7, beta_1 = -0.0, beta_2 = 500.0, delta_1 = 2.0,delta_2 = 2.0
Sensitivity for coefficients in objective:
var = 1, beta_1 = Inf, beta_2 = 3.0, delta_1 = 300.0,delta_2 = 300.0
var = 2, beta_1 = Inf, beta_2 = Inf, delta_1 = 100.0,delta_2 = 100.0
var = 3, beta_1 = -2.0, beta_2 = Inf, delta_1 = 0.0,delta_2 = 0.0
var = 4, beta_1 = Inf, beta_2 = 2.0, delta_1 = 500.0,delta_2 = 500.0
var = 5, beta_1 = -3.0, beta_2 = Inf, delta_1 = 500.0,delta_2 = 500.0
var = 6, beta_1 = Inf, beta_2 = 2.0, delta_1 = 500.0,delta_2 = 500.0
var = 7, beta_1 = -2.0, beta_2 = Inf, delta_1 = 0.0,delta_2 = 0.0
```

Chapter 15

Toolbox API Reference

- *General API conventions.*
- **Command reference:**
  - *Complete list of functions*
  - *mosekopt* - the main interface
  - *Data structures*
- **Optimizer parameters:**
  - *Double*, *Integer*, *String*
  - *Full list*
  - *Browse by topic*
- **Optimizer information items:**
  - *Double*, *Integer*, *Long*
- *Optimizer response codes*
- *Constants*
- *Functions compatible with the MATLAB Optimization Toolbox*
- *Nonlinear API (mskenopt, mskscopt, mskgpopt)*

15.1 API conventions

Problem setup

An optimization problem in Optimization Toolbox for MATLAB is specified using the *prob* structure. The specification of numerical part of the data can be found in Sec. 15.3.1.

Constants

Constants mentioned in Sec. 15.7 and Sec. 15.5 can be used as strings or as symbolic constants. To get the structure with all symbolic constants available execute:

```
[r, res] = mosekopt('symbcon');
```

They can later be used simply as, for example:

```
res.symbcon.MSK_IPAR_OPTIMIZER
```

15.2 Command Reference

The **MOSEK** toolbox provides a set of functions to interface to the **MOSEK** solver.

Main interface

mosekopt is the main interface to **MOSEK**.

Helper functions

These functions provide an easy-to-use but less flexible interface than the *mosekopt* function. They are just wrappers around the *mosekopt* interface written in MATLAB.

- *msklpopt* : Solves linear optimization problems.
- *mskqpopt* : Solves quadratic optimization problems.

Options

Functions for manipulating parameter values.

- *mskoptimget* : Get the solver parameters.
- *mskoptimset* : Set the solver parameters.

MATLAB Optimization Toolbox compatible functions.

Functions that override standard functions from the MATLAB Optimization Toolbox (the user may choose not to install those).

- *linprog* : Solves linear optimization problems.
- *quadprog* : Solves quadratic optimization problems.
- *intlinprog* : Solves linear optimization problems with integer variables.
- *lsqlin* : Solves least-squares with linear constraints.
- *lsqnonneg* : Solves least-squares with non-negativity constraints.

15.2.1 Main Interface

`rcode, res = mosekopt(cmd, prob, param, callback)`

Solves an optimization problem. Data specifying the optimization problem can either be read from a file or be inputted directly from MATLAB. It also makes it possible to write a file and provides other functionalities.

The behavior is specified by the `cmd` parameter which recognizes the following commands:

- `anapro`: Runs the problem analyzer.
- `echo(n)`: Controls how much log information is printed to the screen. `n` must be a nonnegative integer, where 0 means silent. See Sec. 7.3.1.
- `info`: Return the complete task information database in `res.info`. See Sec. 7.5.
- `param`: Return the complete parameter database in `res.param`. See Sec. 7.4.
- `primalrepair`: Performs a primal feasibility repair. See Sec. 14.2.
- `maximize`: Maximize the objective.
- `max` : Sets the objective sense (similar to `maximize`), without performing an optimization.
- `minimize`: Minimize the objective.
- `min`: Sets the objective sense (similar to `minimize`), without performing an optimization.
- `nokeepenv`: Delete the **MOSEK** environment after running the optimizer. This can increase the license checkout overhead significantly and is therefore only intended as a debug feature. Can also be used to delete the environment and release all licenses. See Sec. 10.4.
- `read(name)`: Request that data is read from a file `name`. See Sec. 7.3.4.

- `statuskeys(n)`: Controls the format of status keys (problem status, solution status etc.) in the returned problem:
  - `statuskeys(0)` – all the status keys are returned as strings,
  - `statuskeys(1)` – all the status keys are returned as numeric codes.
- `symbcon`: Return the list of symbolic constants in `res.symbcon`.
- `write(name)`: Write problem to the file `name`. See Sec. 7.3.3.
- `log(name)`: Write solver log output to the file `name`. See Sec. 7.3.1.
- `version`: Return the **MOSEK** version numbers in `res.version`.
- `debug(n)`: Prints debug information including license paths. `n` must be a nonnegative integer which determines how much to print.
- `toconic prob`: Convert a quadratic problem to conic form. See Sec. 9.1.

Parameters
- `cmd` (string) – The commands to be executed. By default it takes the value `minimize`.
- `prob` (*prob*) – A structure containing the problem data. (optional)
- `param` (struct) – A structure specifying **MOSEK** parameters. See Sec. 7.4. (optional)
- `callback` (*callback*) – A MATLAB structure defining call-back data and functions. See Sec. 7.6. (optional)

Return
- `rcode` (*rescode*) – A response code. See also Sec. 7.1.
- `res` (*res*) – A structure containing solutions and other results from the call. See Sec. 7.1.

15.2.2 Helper Functions

`res = msklpopt(c, a, blc, buc, blx, bux, param, cmd)`

Solves a linear optimization problem of the form

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

`blc=[]` and `buc=[]` mean that the lower and upper bounds are $-\infty$ and $+\infty$, respectively. The same interpretation is used for `blx` and `bux`. The value `-inf` is allowed in `blc` and `blx`. Similarly, `inf` is allowed in *buc* and *bux*.

Parameters
- `c [in]` (double[]) – The objective function vector.
- `a [in]` (double[][]) – A (preferably sparse) matrix.
- `blc [in]` (double[]) – Constraints lower bounds.
- `buc [in]` (double[]) – Constraints upper bounds.
- `blx [in]` (double[]) – Variables lower bounds.
- `bux [in]` (double[]) – Variables upper bounds.
- `param [in]` (struct) – **MOSEK** parameters. (optional)
- `cmd [in]` (string) – The command list. See *mosekopt* for a list of available commands. (optional)

Return `res` (*res*) – Solution information.

`res = mskqpopt(q, c, a, blc, buc, blx, bux, param, cmd)`

Solves the optimization problem

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}x^T Q x + c^T x \\ \text{subject to} & l^c \leq Ax \leq u^c, \\ & l^x \leq x \leq u^x. \end{array}$$

blc=[] and buc=[] mean that the lower and upper bounds are $-\infty$ and $+\infty$, respectively. The same interpretation is used for blx and bux. The value -inf is allowed in blc and blx. Similarly, inf is allowed in *buc* and *bux*.

Parameters

- q (double[]) – The matrix Q, which must be symmetric positive semidefinite.
- c [in] (double[]) – The objective function vector.
- a (double[][]) – A (preferably) sparse matrix.
- blc [in] (double[]) – Constraints lower bounds.
- buc [in] (double[]) – Constraints upper bounds
- blx [in] (double[]) – Variables lower bounds
- bux [in] (double[]) – Variables upper bounds
- param [in] (struct) – **MOSEK** parameters. (optional)
- cmd [in] (string) – The command list. See *mosekopt* for a list of available commands. (optional)

Return res (*res*) – Solution information.

15.2.3 Options

val = mskoptimget(options, param, default)

Obtains the value of an optimization parameter. See the *mskoptimset* function for which parameters that can be set.

Parameters

- options [in] (struct) – The optimization options structure.
- param [in] (string) – Name of the optimization parameter for which the value should be obtained.
- default [in] (string) – If param is not defined, the value of default is returned instead. (optional)

Return val (list) – Value of the required option. If the option does not exist, then [] is returned unless the value default is defined in which case the default value is returned.

options = mskoptimset(arg1, arg2, param1, value1, param2, value2, ...)

Obtains and modifies the optimization options structure. Only a subset of the fields in the optimization structure recognized by the MATLAB Optimization Toolbox is recognized by **MOSEK**. In addition the optimization options structure can be used to modify all the **MOSEK** specific parameters defined in Sec. 15.5.

- .Diagnostics Used to control how much diagnostic information is printed. Following values are accepted:

off	No diagnostic information is printed.
on	Diagnostic information is printed.

- .Display Defines what information is displayed. The following values are accepted:

off	No output is displayed.
iter	Some output is displayed for each iteration.
final	Only the final output is displayed.

- .MaxIter Maximum number of iterations allowed.
- .Write A file name to write the problem to. If equal to the empty string no file is written. E.g the option Write(myfile.opf) writes the file myfile.opf in the opf format.

Parameters

- arg1 [in] (None) – Is allowed to be any of the following two things (optional):
  – Any string — The same as using no argument.

– A structure — The argument is assumed to be a structure containing options, which are copied to the return options.

- `param1 [in]` (string) – A string containing the name of a parameter that should be modified. (optional)
- `value1 [in]` (None) – The new value assigned to the parameter with the name `param1`. (optional)
- `param2 [in]` (None) – See `param1`. (optional)
- `value2 [in]` (None) – See `value1`. (optional)

Return `options` (struct) – The updated optimization options structure.

15.2.4 MATLAB Optimization Toolbox Compatible Functions.

`x`, `fval`, `exitflag`, `output` = `intlinprog(f, intcon, A, b, B, c, l, u, options)`
`x`, `fval`, `exitflag`, `output` = `intlinprog(problem)`

Solves the mixed-integer linear optimization problem:

$$\begin{array}{ll} \text{minimize} & f^T x \\ \text{subject to} & Ax \le b, \\ & Bx = c, \\ & l \le x \le u, \\ & x(\text{intcon}) \in \mathbb{Z}. \end{array}$$

Parameters

- `f [in]` (double[]) – The objective function.
- `intcon [in]` (int[]) – The list of variables constrained to the set $\mathbb{Z}$.
- `A [in]` (double[][]) – Constraint matrix for the inequalities. Use `A=[]` if there are no inequalities.
- `b [in]` (double[]) – Right-hand side for the inequalities. Use `b=[]` if there are no inequalities.
- `B [in]` (double[][]) – Constraint matrix for the equalities. (optional)
- `c [in]` (double[]) – Right-hand side for the equalities. (optional)
- `l [in]` (double[]) – Lower bounds for variables. Use `-inf` to represent infinite lower bounds. (optional)
- `u [in]` (double[]) – Upper bounds for variables. Use `inf` to represent infinite upper bounds. (optional)
- `options [in]` (struct) – An optimization options structure. See the *`mskoptimset`* function for the definition of the optimization options structure (optional). This function uses the options
  – `.Diagnostics`
  – `.Display`
  – `.MaxTime` Time limit in seconds.
  – `.MaxNodes` The maximum number of branch-and-bounds allowed.
  – `.Write` Name of file to save the problem.
- `problem [in]` (struct) – A structure containing the fields `f, intcon, A, b, B, c, l, u` and `options`.

Return

- `x` (double[]) – The solution x.
- `fval` (double) – The objective $f^T x$.
- `exitflag` (int) – A number which has the interpretation:
  – 1 The function returned an integer feasible solution.
  – −2 The problem is infeasible.
  – −4 `MaxNodes` reached without converging.
  – −5 `MaxTime` reached without converging.

`x`, `fval`, `exitflag`, `output`, `lambda` = `linprog(f, A, b, B, c, l, u, options)` `x, fval, exitflag, output, lambda = linprog(problem)`

Solves the linear optimization problem:

$$\begin{array}{ll} \text{minimize} & f^T x \\ \text{subject to} & Ax \leq b, \\ & Bx = c, \\ & l \leq x \leq u. \end{array}$$

Parameters

- `f [in]` (double[]) – The objective function.
- `A [in]` (double[][]) – Constraint matrix for the inequalities. Use $A = []$ if there are no inequalities.
- `b [in]` (double[]) – Right-hand side for the inequalities. Use $b = []$ if there are no inequalities.
- `B [in]` (double[][]) – Constraint matrix for the equalities. (optional)
- `c [in]` (double[]) – Right-hand side for the equalities. (optional)
- `l [in]` (double[]) – Lower bounds on the variables. Use `-inf` to represent infinite lower bounds. (optional)
- `u [in]` (double[]) – Upper bounds on the variables. Use `inf` to represent infinite upper bounds. (optional)
- `options [in]` (struct) – An optimization options structure (optional). See the *mskoptimset* function for the definition of the optimization options structure. This function uses the options
  - `.Diagnostics`
  - `.Display`
  - `.MaxIter`
  - `.Simplex` Choose the simplex algorithm: `'on'` — the optimizer chooses wither primal or dual simplex (as in *"MSK_OPTIMIZER_FREE_SIMPLEX"*), `'primal'` — use primal simplex, `'dual'` — use dual simplex. The `'primal'` and `'dual'` values are specific for the **MOSEK** interface, and not present in the standard MATLAB version.
  - `.Write` Name of file to save the problem.
- `problem [in]` (struct) – structure containing the fields `f`, `A`, `b`, `B`, `c`, `l`, `u` and `options`.
- `output [in]` (struct) – A structure with the following fields
  - `.iterations` Number of interior-point iterations spent to reach the optimum.
  - `.algorithm` Always defined as `'MOSEK'`.
- `lambda [in]` (struct) – A struct with the following fields
  - `.lower` Lagrange multipliers for lower bounds l.
  - `.upper` Lagrange multipliers for upper bounds u.
  - `.ineqlin` Lagrange multipliers for the inequalities.
  - `.eqlin` Lagrange multipliers for the equalities.

Return

- `x` (double[]) – The optimal x solution.
- `fval` (double) – The optimal objective value, i.e. $f^T x$.
- `exitflag` (int) – A number which has the interpretation [in]:
  - < 0 The problem is likely to be either primal or dual infeasible.
  - $= 0$ The maximum number of iterations was reached.
  - > 0 x is an optimal solution.

`x, resnorm, residual, exitflag, output, lambda = lsqlin(C, d, A, b, B, c, l, u, x0, options)`

Solves the linear least squares problem:

$$\begin{array}{ll} \text{minimize} & \frac{1}{2} \|Cx - d\|_2^2 \\ \text{subject to} & Ax \leq b, \\ & Bx = c, \\ & l \leq x \leq u. \end{array}$$

Parameters

- `C [in]` (double[][]) – The matrix in the objective.
- `d [in]` (double[]) – The vector in the objective.
- `A [in]` (double[][]) – Constraint matrix for the inequalities. Use $A = []$ if there are no inequalities.
- `b [in]` (double[]) – Right-hand side for the inequalities. Use $b = []$ if there are no inequalities.
- `B [in]` (double[][]) – Constraint matrix for the equalities. (optional)
- `c [in]` (double[]) – Right-hand side for the equalities. (optional)
- `l [in]` (double[]) – Lower bounds on the variables. Use `-inf` to represent infinite lower bounds. (optional)
- `u [in]` (double[]) – Upper bounds on the variables. Use `inf` to represent infinite upper bounds. (optional)
- `x0 [in]` (double[]) – Ignored by **MOSEK**. (optional)
- `options [in]` (struct) – An optimization options structure (optional). See the function *mskoptimset* function for the definition of the optimization options structure. This function uses the options
  - `.Diagnostics`
  - `.Display`
  - `.MaxIter`
  - `.Write`

Return

- `x` (double[]) – The optimal x solution.
- `resnorm` (double) – The squared norm of the optimal residuals, i.e. $\|Cx - d\|_2^2$ evaluated at the optimal solution.
- `residual` (double) – The residual $Cx - d$.
- `exitflag` (int) – A scalar which has the interpretation:
  - < 0 The problem is likely to be either primal or dual infeasible.
  - $= 0$ The maximum number of iterations was reached.
  - > 0 x is the optimal solution.
- `output` (struct) –
  - `.iterations` Number of iterations spent to reach the optimum.
  - `.algorithm` Always defined as `'MOSEK'`.
- `lambda` (struct) –
  - `.lower` Lagrange multipliers for lower bounds l.
  - `.upper` Lagrange multipliers for upper bounds u.
  - `.ineqlin` Lagrange multipliers for inequalities.
  - `.eqlin` Lagrange multipliers for equalities.

`x, resnorm, residual, exitflag, output, lambda = lsqnonneg(C, d, x0, options)`

Solves the linear least squares problem:

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\|Cx - d\|_2^2 \\ \text{subject to} & x \geq 0. \end{array}$$

Parameters

- `C [in]` (double[][]) – The matrix in the objective.
- `d [in]` (double[]) – The vector in the objective.
- `x0 [in]` (double[]) – Ignored by **MOSEK**. (optional)
- `options [in]` (struct) – An optimization options structure (optional). See the *mskoptimset* function for the definition of the optimization options structure. This function uses the options
  - `.Diagnostics`

- .Display
- .MaxIter
- .Write

Return

- x (double[]) – The x solution.
- resnorm (double) – The squared norm of the optimal residuals, i.e. $\|Cx - d\|_2^2$ evaluated at the optimal solution.
- exitflag (int) – A number which has the interpretation:
  - < 0 The problem is likely to be either primal or dual infeasible.
  - $= 0$ The maximum number of iterations was reached.
  - > 0 x is optimal solution.
- output (struct) –
  - .iterations Number of iterations spend to reach the optimum.
  - .algorithm Always defined to be 'MOSEK'.
- lambda (struct) –
  - .lower Lagrange multipliers for lower bounds l.
  - .upper Lagrange multipliers for upper bounds u.
  - .ineqlin Lagrange multipliers for inequalities.
  - .eqlin Lagrange multipliers for equalities.

x, fval, exitflag, output, lambda = quadprog(H, f, A, b, B, c, l, u, x0, options)

Solves the quadratic optimization problem:

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}x^T Hx + f^T x \\ \text{subject to} & Ax \leq b, \\ & Bx = c, \\ & l \leq x \leq u. \end{array}$$

Parameters

- H [in] (double[][]) – Hessian of the objective function. The matrix H must be symmetric positive semidefinite. Contrary to the MATLAB optimization toolbox, **MOSEK** handles only the cases where H is positive semidefinite. On the other hand **MOSEK** always computes a global optimum.
- f [in] (double[]) – The linear term of the objective.
- A [in] (double[][]) – Constraint matrix for the inequalities. Use $A = []$ if there are no inequalities.
- b [in] (double[]) – Right-hand side for the inequalities. Use $b = []$ if there are no inequalities.
- B [in] (double[][]) – Constraint matrix for the equalities. (optional)
- c [in] (double[]) – Right-hand side for the equalities. (optional)
- l [in] (double[]) – Lower bounds on the variables. Use -inf to represent infinite lower bounds. (optional)
- u [in] (double[]) – Upper bounds on the variables. Use inf to represent infinite upper bounds. (optional)
- x0 [in] (double[]) – Ignored by **MOSEK**. (optional)
- options [in] (struct) – An optimization options structure (optional). See the *mskoptimset* function for the definition of the optimizations options structure. This function uses the options
  - .Diagnostics
  - .Display
  - .MaxIter
  - .Write

Return

- x (double[]) – The x solution.

- `fval` (double) – The optimal objective value i.e. $\frac{1}{2}x^T Hx + f^T x$.
- `exitflag` (int) – A scalar which has the interpretation:
  - < 0 The problem is likely to be either primal or dual infeasible.
  - $= 0$ The maximum number of iterations was reached.
  - > 0 x is an optimal solution.
- `output` (struct) – A structure with the following fields
  - `.iterations` Number of iterations spent to reach the optimum.
  - `.algorithm` Always defined as `'MOSEK'`.
- `lambda` (struct) – A structure with the following fields
  - `.lower` Lagrange multipliers for lower bounds l.
  - `.upper` Lagrange multipliers for upper bounds u.
  - `.ineqlin` Lagrange multipliers for inequalities.
  - `.eqlin` Lagrange multipliers for equalities.

15.3 Data Structures and Notation

We specify the notation and data structures used in the interface.

Problem definition

- *prob* — describes an optimization problem.
- *cones* — description of cones.
- *names* — names of objects in the optimization problem.
- *barc*, *bara* — description of the semidefinite part.

Solutions

- *res* — result returned by *mosekopt*.
- *solver_solutions* — solutions.
- *solution* — one solution.

Other

- *primal_repair* — used in feasibility repair.
- *prisen*, *prisen_data*, *duasen* — used in sensitivity analysis.
- *callback* — used to set up a callback function.

15.3.1 Notation

Linear problem

A linear problem has the form:

$$
\begin{array}{lrcccll}
\text{minimize} & & & \sum_{j=1}^{n} c_j x_j + c^f & & & \\
\text{subject to} & l_i^c & \leq & \sum_{j=1}^{n} a_{ij} x_j & \leq & u_i^c, & i = 1, \ldots, m, \\
 & l_j^x & \leq & x_j & \leq & u_j^x, & j = 1, \ldots, n.
\end{array}
\tag{15.1}
$$

It has n variables and m linear constraints. See Sec. 12.1.

Conic problem

A conic problem is an extension of a linear problem and has the form:

$$
\begin{array}{lccccll}
\text{minimize} & & & \sum_{j=1}^{n} c_j x_j + \sum_{j=1}^{p} \langle \overline{C}_j, \overline{X}_j \rangle + c^f & & & \\
\text{subject to} & l_i^c & \leq & \sum_{j=1}^{n} a_{ij} x_j + \sum_{j=1}^{p} \langle \overline{A}_{ij}, \overline{X}_j \rangle & \leq & u_i^c, & i = 1, \ldots, m, \\
 & l_j^x & \leq & x_j & \leq & u_j^x, & j = 1, \ldots, n, \\
 & & & x \in \mathcal{K}, & & & \\
 & & & \overline{X}_j \in \mathcal{S}_+^{r_j}, & & & j = 1, \ldots, p
\end{array}
\tag{15.2}
$$

It has n variables, m linear constraints, a cone $\mathcal{K}$ (see below) and $p \geq 0$ semidefinite variables. See also Sec. 12.2 (or Sec. 12.3 for SDP). The conic constraint

$$x \in \mathcal{K}$$

means that a partitioning of x belongs to a set of cones $\mathcal{K} = \mathcal{K}_1 \times \ldots \times \mathcal{K}_s$, which can be one of:

- Quadratic cone

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

- Rotated quadratic cone

$$\mathcal{Q}_r^n = \left\{ x \in \mathbb{R}^n : 2x_1 x_2 \geq \sum_{j=3}^{n} x_j^2, \quad x_1 \geq 0, \quad x_2 \geq 0 \right\}.$$

- Primal exponential cone

$$K_{\exp} = \left\{ x \in \mathbb{R}^3 : x_1 \geq x_2 \exp(x_3/x_2), \quad x_1, x_2 \geq 0 \right\}$$

as well as its dual

$$K_{\exp}^* = \left\{ x \in \mathbb{R}^3 : x_1 \geq -x_3 e^{-1} \exp(x_2/x_3), \quad x_3 \leq 0, x_1 \geq 0 \right\}.$$

- Primal power cone (with parameter $0 < \alpha < 1$)

$$\mathcal{P}_n^{\alpha, 1-\alpha} = \left\{ x \in \mathbb{R}^n : x_1^\alpha x_2^{1-\alpha} \geq \sqrt{\sum_{j=3}^{n} x_j^2}, \quad x_1, x_2 \geq 0 \right\}$$

as well as its dual

$$(\mathcal{P}_n^{\alpha, 1-\alpha})^* = \left\{ x \in \mathbb{R}^n : \left(\frac{x_1}{\alpha}\right)^\alpha \left(\frac{x_2}{1-\alpha}\right)^{1-\alpha} \geq \sqrt{\sum_{j=3}^{n} x_j^2}, \quad x_1, x_2 \geq 0 \right\}.$$

- The set $\mathbb{R}^n$.
- The zero cone $\{(0, \ldots, 0)\}$.

Membership in the trivial cones does not have to be specified.

Conic problem with affine conic constraints

A conic problem with affine conic constraints is an extension of a linear problem and has the form:

$$
\begin{array}{lrcccll}
\text{minimize} & & & \sum_{j=1}^{n} c_j x_j + \sum_{j=1}^{p} \left\langle \overline{C}_j, \overline{X}_j \right\rangle + c^f & & & \\
\text{subject to} & l_i^c & \leq & \sum_{j=1}^{n} a_{ij} x_j + \sum_{j=1}^{p} \left\langle \overline{A}_{ij}, \overline{X}_j \right\rangle & \leq & u_i^c, & i = 1, \ldots, m, \\
 & l_j^x & \leq & x_j & \leq & u_j^x, & j = 1, \ldots, n, \\
 & & & Fx + g \in \mathcal{K}, & & & \\
 & & & \overline{X}_j \in \mathcal{S}_+^{r_j}, & & & j = 1, \ldots, p
\end{array}
\tag{15.3}
$$

It has n variables, m linear constraints, a cone $\mathcal{K}$ (see below) and $p \geq 0$ semidefinite variables. See also Sec. 12.5.

The conic constraint

$$Fx + g \in \mathcal{K}$$

where $F \in \mathbb{R}^{k \times n}$ and $g \in \mathbb{R}^k$ means that an affine combination of x belongs to a product of cones $\mathcal{K} = \mathcal{K}_1 \times \ldots \times \mathcal{K}_s$ of total length (dimension) k. The available cone types are the same as above.

Quadratic and quadratically constrained problems

A problem with quadratic objective or constraints has the form:

$$
\begin{array}{lrcccll}
\text{minimize} & & & \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} q_{ij}^o x_i x_j + \sum_{j=1}^{n} c_j x_j + c^f & & & \\
\text{subject to} & l_i^c & \leq & \frac{1}{2} \sum_{j=1}^{n} \sum_{k=1}^{n} q_{jk}^i x_j x_k + \sum_{j=1}^{n} a_{ij} x_j & \leq & u_i^c, & i = 1, \ldots, m, \\
 & l_j^x & \leq & x & \leq & u_j^x, & j = 1, \ldots, n.
\end{array}
\tag{15.4}
$$

It has n variables, and m constraints. The matrix $Q^o = (q_{ij}^o)_{i=1,\ldots,n, j=1,\ldots,n}$ must be symmetric positive semidefinite. See also Sec. 12.4. Each of the matrices $Q^i = (q_{jk}^i)_{j=1,\ldots,n, k=1,\ldots,n}$ for $j = 1 \ldots, m$ must be

- negative semidefinite if $-\infty < l_i^c$, $u_i^c = +\infty$,
- positive semidefinite if $-\infty = l_i^c$, $u_i^c < +\infty$,
- zero otherwise.

Mixed-integer problems

All problems without semidefinite variables may be integer-constrained, i.e., for some set $\mathcal{J} \subseteq \{1, \ldots, n\}$ we require

$$x_j \in \mathbb{Z} \text{ for all } j \in \mathcal{J} \tag{15.5}$$

Minimization vs. Maximization

The objective of every problem can be maximized rather than minimized without any change. In case of quadratic problems the matrix Q^o must be negative semidefinite.

Data specification in MATLAB

- The linear constraint matrix $A = (a_{ij})_{i=1\ldots,m, j=1,\ldots,n}$ must be a sparse matrix. The dimensions of A are used to determine the number of constraints m and the number of variables n in the problem.
- The symmetric matrices Q^o, Q^i, $\overline{C}_j$ and $\overline{A}_{ij}$ are specified in sparse triplet format discarding zero elements, and since they are symmetric, only the lower triangular parts should be specified. A generic matrix M specified in sparse triplet format is given by three arrays `subi`, `subj` and `val` of the same length such that

$$M_{\texttt{subi[t]},\texttt{subj[t]}} = \texttt{val}[\texttt{t}], \qquad t = 1, \ldots, \texttt{len}(\texttt{val})$$

- For a specification of the cones $\mathcal{K}$ see *cones*.

- For a specification of the semidefinite part see *bara* and *barc*.

The parameters of the optimization problem are stored using one or more subfields of the *prob* structure using the naming convention in Table 15.1. Only `a` is obligatory. All other fields are optional depending on what problem type is defined.

Table 15.1: The relation between fields and problem parameters

Field name	Type	Dimension	Problem parameters
a	sparse matrix	$m \times n$	a_{ij}
c	double[]	n	c_j
cfix	double	1	c^f
blc	double[]	m	l_i^c
buc	double[]	m	u_i^c
blx	double[]	n	l_j^x
bux	double[]	n	u_j^x
ints.subs	int[]	$\|\mathcal{J}\|$	$\mathcal{J}$
cones	*cones*		$\mathcal{K}$
f	sparse matrix	$k \times n$	F
g	double[]	k	g
bardim	int[]	p	r_j
barc	*barc*		$\overline{C}_j$
bara	*bara*		$\overline{A}_{ij}$
qosubi	int[]	len(qoval)	q_{ij}^o, sparse rep.
qosubj	int[]	len(qoval)	q_{ij}^o, sparse rep.
qoval	double[]	len(qoval)	q_{ij}^o, sparse rep.
qcsubk	int[]	len(qcval)	q_{ij}^k, sparse rep.
qcsubi	int[]	len(qcval)	q_{ij}^k, sparse rep.
qcsubj	int[]	len(qcval)	q_{ij}^k, sparse rep.
qcval	double[]	len(qcval)	q_{ij}^k, sparse rep.

The `int` type indicates that the field must contain an integer value, `double` indicates any real number. This distinction is only a convenience for the reader — all actual data structures in MATLAB are ordinary matrices/arrays of floating-point numbers.

The sparse representation of quadratic terms is:

$$\begin{aligned} q^o_{\texttt{qosubi(t)},\texttt{qoval(t)}} &= \texttt{qoval(t)}, \quad t = 1, 2, \ldots, \texttt{length}(\texttt{qoval}), \\ q^{\texttt{qcsubk(t)}}_{\texttt{qcsubi(t)},\texttt{qcsubj(t)}} &= \texttt{qcval(t)}, \quad t = 1, 2, \ldots, \texttt{length}(\texttt{qcval}). \end{aligned} \tag{15.6}$$

Since Q^o, Q^i are by assumption symmetric, all elements are assumed to belong to the lower triangular part. If an element is specified multiple times, the different elements are added together.

15.3.2 Data Types and Structures

`prob`

The `prob` data structure is used to communicate an optimization problem to **MOSEK** or for **MOSEK** to return an optimization problem to the user. It defines an optimization problem using a number of subfields. Most of the fields are optional, depending on what problem type is being solved.

Fields

- `names` (*names*) – A structure which contains the names of the problem, variables, constraints and so on.
- `a` (double[][]) – The linear constraint matrix. It is obligatory, and its dimensions define the number of variables and constraints. It must be a **sparse matrix**. This field should always be defined, even if the problem does not have any constraints. In that case a sparse matrix having zero rows and the correct number of columns is the appropriate definition of the field.
- `c` (double[]) – Linear term in the objective.

- `cfix` (double) – Fixed term in the objective.
- `blc` (double[]) – Lower bounds of the constraints. $-\infty$ denotes an infinite lower bound. If the field is not defined or `blc==[]`, then all the lower bounds are assumed to be equal to $-\infty$.
- `buc` (double[]) – Upper bounds of the constraints. ∞ denotes an infinite upper bound. If the field is not defined or `buc==[]`, then all the upper bounds are assumed to be equal to ∞.
- `blx` (double[]) – Lower bounds on the variables. $-\infty$ denotes an infinite lower bound. If the field is not defined or `blx==[]`, then all the lower bounds are assumed to be equal to $-\infty$.
- `bux` (double[]) – Upper bounds on the variables. ∞ denotes an infinite upper bound. If the field is not defined or `bux==[]`, then all the upper bounds are assumed to be equal to ∞.
- `bardim` (int[]) – A list with the dimensions of the semidefinite variables.
- `barc` (*barc*) – A structure for specifying $\overline{C}_j$.
- `bara` (*bara*) – A structure for specifying $\overline{A}_{ij}$.
- `qosubi` (int[]) – i subscripts in the sparse specification of q^o_{ij} in Q^o. See (15.6).
- `qosubj` (int[]) – j subscripts in the sparse specification of q^o_{ij} in Q^o. See (15.6).
- `qoval` (double[]) – Numerical values in the sparse specification of q^o_{ij} in Q^o. See (15.6).
- `qcsubk` (int[]) – k subscripts in the sparse specification of q^k_{ij} in Q^k. See (15.6)
- `qcsubi` (int[]) – i subscripts in the sparse specification of q^k_{ij} in Q^k. See (15.6)
- `qcsubj` (double[]) – j subscripts in the sparse specification of q^k_{ij} in Q^k. See (15.6)
- `qcval` (double[]) – Numerical values in the sparse specification of q^k_{ij} in Q^k. See (15.6)
- `f` (double[][]) – The matrix of affine conic constraints. It must be a sparse matrix.
- `g` (double[]) – The constant term of affine conic constraints. If not present or `g==[]` it is assumed $g = 0$.
- `ints.sub` (int[]) – A list of indexes of integer-constrained variables. `ints.sub` is identical to the set $|$ in (15.5).
- `cones` (*cones*) – A structure defining either the conic constraints in (15.2) or the affine conic constraints in (15.3).
- `sol` (*solver_solutions*) – A structure containing a guess on the optimal solution which some of the optimizers in **MOSEK** may exploit.
- `primlarepair` (*primal_repair*) – Specification of primal feasibility repair. See Sec. 14.2.1.
- `prisen` (*prisen*) – Request sensitivity analysis. See Sec. 14.3.
- `duasen` (*duasen*) – Request sensitivity analysis. See Sec. 14.3.

`res`

Contains a response from *mosekopt*.

Fields

- `sol` (*solver_solutions*) – A structure containing solutions (if any).
- `info` (struct) – A structure containing information items (if requested by the command `info` in *mosekopt*). See Sec. 7.5.
- `prob` (*prob*) – Contains the problem data, if the command `read` was used to read a problem from a file. See Sec. 7.3.4.
- `param` (struct) – A structure which contains the complete **MOSEK** parameter database (if requested by the command `param` in *mosekopt*). See Sec. 7.4.
- `symbcon` (struct) – A structure which contains symbolic constants and their numerical values (if requested by the command `symbcon` in *mosekopt*). See Sec. 15.7 and Sec. 15.6.

- `version` (struct) – A structure which contains the **MOSEK** version numbers (if requested by the command `version` in *mosekopt*).
- `prisen` (*prisen*) – A structure with results of sensitivity analysis (if requested by passing `prisen` data in `prob`). See Sec. 14.3.
- `duasen` (*duasen*) – A structure with results of sensitivity analysis (if requested by passing `duasen` data in `prob`). See Sec. 14.3.

`solver_solutions`

It contains informations about initial/final solutions. Availability of solutions depends on the problem/algorithm type, see Sec. 7.1.2.

Fields

- `itr` (*solution*) – Interior solution.
- `bas` (*solution*) – Basic solution.
- `int` (*solution*) – Integer solution.

`solution`

Stores information about one solution. See Sec. 7.1.2.

Fields

- `prosta` (string) – Problem status (*prosta*).
- `solsta` (string) – Solution status (*solsta*).
- `skc` (string[]) – Linear constraint status keys (*stakey*).
- `skx` (string[]) – Variable status keys (*stakey*).
- `skn` (string[]) – Conic constraint status keys (*stakey*, not in basic solution).
- `xc` (double[]) – Constraint activities, i.e., $x_c = Ax$ where x is the optimal solution.
- `xx` (double[]) – The optimal x solution.
- `barx` (list) – Semidefinite variable solution (not in basic solution).
- `y` (double[]) – Identical to `sol.slc-sol.suc` (not in integer solution).
- `slc` (double[]) – Dual variable for constraint lower bounds (not in integer solution).
- `suc` (double[]) – Dual variable for constraint upper bounds (not in integer solution).
- `slx` (double[]) – Dual variable for variable lower bounds (not in integer solution).
- `sux` (double[]) – Dual variable for variable upper bounds (not in integer solution).
- `snx` (double[]) – Dual variable of conic constraints (not in basic or integer solution).
- `doty` (double[]) – Dual variables of affine conic constraints (not in basic or integer solution).
- `bars` (list) – Dual variable of semidefinite domains (not in basic or integer solution).
- `pobjval` (double) – The primal objective value.
- `dobjval` (double) – The dual objective value (not in integer solution).

`names`

This structure is used to store all the names of individual items in the optimization problem such as the constraints and the variables.

Fields

- `name` (string) – contains the problem name.
- `obj` (string) – contains the name of the objective.
- `con` (cell) – a cell array where `names.con{i}` contains the name of the i-th constraint.
- `var` (cell) – a cell array where `names.var{j}` contains the name of the j-th variable.

- cone (cell) – a cell array where names.cone{t} contains the name of the t-th conic constraint.
- barvar (cell) – a cell array where names.barvar{j} contains the name of the j-th semidefinite variable.

cones

Represents either conic constraints or affine conic constraints.

Conic constraints. For conic constraints $x \in \mathcal{K}$, where $\mathcal{K} = \mathcal{K}_1 \times \cdots \times \mathcal{K}_s$, cones is a structure containing three or four fields:

- type (list) — An array with the cone type for each cone; see *conetype*.
- sub (int[]) — A concatenation of index lists of all the cones.
- subptr (int[]) — An array of pointers indicating the beginning of consecutive cones in sub.
- conepar (double[]) — An array of cone parameters; for a power cone this is the exponent α, for other cone types the value is irrelevant. If the problem contains no power cones this field is not required.

The arrays type and sub (and conepar if present) must have the same length s. For every $j = 1, \ldots, s$ the vector $x_{\texttt{sub[subptr[j]:(subptr[j+1]-1)]}}$ belongs to a cone of type type[j] with parameter conepar[j]. For example the two cones

$$x_5 \geq \sqrt{x_3^2 + x_1^2}, \quad x_6^{0.3} x_4^{0.7} \geq \sqrt{x_2^2 + x_7^2}$$

```
cones.type   = [res.symbcon.MSK_CT_QUAD, res.symbcon.MSK_CT_PPOW];
cones.sub    = [5, 3, 1, 6, 4, 2, 7];
cones.subptr = [1, 4];
cones.conepar= [0.0, 0.3];
```

Affine conic constraints. For affine conic constraints $Fx + g \in \mathcal{K}$, where $\mathcal{K} = \mathcal{K}_1 \times \cdots \times \mathcal{K}_s$, cones is a list consisting of s concatenated cone descriptions. If a cone requires no additional parameters (quadratic, rotated quadratic, exponential, zero) then its description is

$$[\text{type}, \text{len}]$$

where type is the type (*conetype*) and len is the length (dimension). The length must be present. If a cone requires additional parameters (power cones) then its description has the form

$$[\text{type}, \text{len}, k, a_1, \ldots, a_k]$$

where $k \geq 1$ is an integer and $a_1, \ldots, a_k \in \mathbb{R}$ are the additional parameters. For the power cone it is required that $k = 2$ and then a power cone with parameter $\alpha = a_1/(a_1 + a_2)$ is constructed. For example $\mathcal{Q}^5 \times \mathcal{P}_4^{0.4}$ could be defined as

```
cones = [res.symbcon.MSK_CT_QUAD 5 res.symbcon.MSK_CT_PPOW, 4, 2, 40.0, 60.0 ];
```

barc

Together with field bardim this structure specifies the symmetric matrices $\overline{C}_j$ in the objective for semidefinite problems.

The symmetric matrices are specified in block-triplet format as

$$[\overline{C}_{\texttt{barc.subj(t)}}]_{\texttt{barc.subk(t)},\texttt{barc.subl(t)}} = \texttt{barc.val(t)}, \quad t = 1, 2, \ldots, \texttt{length(barc.subj)}.$$

Only the lower triangular parts of $\overline{C}_j$ are specified, i.e., it is required that

$$\texttt{barc.subk(t)} \geq \texttt{barc.subl(t)}, \quad t = 1, 2, \ldots, \texttt{length(barc.subk)},$$

and that

$$1 \leq \texttt{barc.subk(t)} \leq \texttt{bardim(barc.subj(t))}, \quad t = 1, 2, \ldots, \texttt{length(barc.subj)}.,$$

All the structure fields must be arrays of the same length.

Fields

- subj (int[]) – Semidefinite variable indices j.
- subk (int[]) – k subscripts of nonzeros elements.
- subl (int[]) – l subscripts of nonzeros elements.
- val (double) – Numerical values.

bara

Together with the field bardim this structure specifies the symmetric matrices $\overline{A}_{ij}$ in the constraints of semidefinite problems.

The symmetric matrices are specified in block-triplet format as

$$[\overline{A}_{\mathtt{bara.subi(t)},\mathtt{bara.subj(t)}}]_{\mathtt{bara.subk(t)},\mathtt{bara.subl(t)}} = \mathtt{bara.val(t)}, \quad t = 1, 2, \ldots, \mathtt{length(bara.subi)}.$$

Only the lower triangular parts of $\overline{A}_{ij}$ are specified, i.e., it is required that

$$\mathtt{bara.subk(t)} \geq \mathtt{bara.subl(t)}, \quad t = 1, 2, \ldots, \mathtt{length(bara.subk)},$$

and that

$$1 \leq \mathtt{bara.subk(t)} \leq \mathtt{bardim(bara.subj(t))}, \quad t = 1, 2, \ldots, \mathtt{length(bara.subj)},$$

All the structure fields must be arrays of the same length.

Fields

- subi (int[]) – Constraint indices i.
- subj (int[]) – Semidefinite variable indices j.
- subk (int[]) – k subscripts of nonzeros elements.
- subl (int[]) – l subscripts of nonzeros elements.
- val (double[]) – Numerical values.

primal_repair

A structure holding data for primal feasibility repair. If either of the subfields is missing, it assumed to be a vector with value 1 of appropriate dimension. See Sec. 14.2.1.

Fields

- wlc (double[]) – Weights for lower bounds on constraints.
- wuc (double[]) – Weights for upper bounds on constraints.
- wlx (double[]) – Weights for lower bounds on variables.
- wux (double[]) – Weights for upper bounds on variables.

prisen

A structure holding information about primal sensitivity analysis. See Sec. 14.3.

Fields

- cons (*prisen_data*) – Constraints shadow prices.
- vars (*prisen_data*) – Variables shadow prices.

prisen_data

A structure holding information about shadow prices of constraints or variables.

Fields

- subl (int[]) – Indices of variables/constraints to be analyzed for lower bounds.
- subu (int[]) – Indices of variables/constraints to be analyzed for upper bounds.
- lr_bl (double[]) – Left value β_1 in the linearity interval for a lower bound.
- rr_bl (double[]) – Right value β_2 in the linearity interval for a lower bound.
- ls_bl (double[]) – Left shadow price s_l for a lower bound.
- rs_bl (double[]) – Right shadow price s_r for a lower bound.
- lr_bu (double[]) – Left value β_1 in the linearity interval for an upper bound.

- `rr_bu` (double[]) – Right value β_2 in the linearity interval for an upper bound.
- `ls_bu` (double[]) – Left shadow price s_l for an upper bound.
- `rs_bu` (double[]) – Right shadow price s_r for an upper bound.

`duasen`

A structure holding information about dual sensitivity analysis. See Sec. 14.3.

Fields

- `sub` (int[]) – Indices of variables to be analyzed.
- `lr_c` (double) – Left value β_1 in linearity interval for an objective coefficient
- `rr_c` (double) – Right value β_2 in linearity interval for an objective coefficient
- `ls_c` (double) – Left shadow price s_l for an objective coefficient
- `rs_c` (double) – Right shadow price s_r for an objective coefficient

`callback`

A structure containing callback information (all subfields are optional).

Fields

- `loghandle` (struct) – A data structure or just `[]`.
- `log` (string) – Log handler. The name of a user-defined function which must accept two input arguments, e.g.,

```
function myprint(handle,str)
```

  where `handle` will be identical to `callback.handle` when `myfunc` is called, and `str` is a string of text from the log.
- `iterhandle` (struct) – A data structure or just `[]`.
- `iter` (string) – Progress callback handler. The name of a user-defined function which must accept three input arguments,

```
function [r] = callback_handler(handle,where,info)
```

  where `handle` will be identical to `callback.iterhandle` when the handler is called, `where` indicates the current progress of the solver (*callback*) and `info` is the current information items list. See Sec. 7.6 for further details.

15.4 Parameters grouped by topic

Analysis

- *MSK_DPAR_ANA_SOL_INFEAS_TOL*
- *MSK_IPAR_ANA_SOL_BASIS*
- *MSK_IPAR_ANA_SOL_PRINT_VIOLATED*
- *MSK_IPAR_LOG_ANA_PRO*

Basis identification

- *MSK_DPAR_SIM_LU_TOL_REL_PIV*
- *MSK_IPAR_BI_CLEAN_OPTIMIZER*
- *MSK_IPAR_BI_IGNORE_MAX_ITER*
- *MSK_IPAR_BI_IGNORE_NUM_ERROR*
- *MSK_IPAR_BI_MAX_ITERATIONS*
- *MSK_IPAR_INTPNT_BASIS*
- *MSK_IPAR_LOG_BI*
- *MSK_IPAR_LOG_BI_FREQ*

Conic interior-point method

- MSK_DPAR_INTPNT_CO_TOL_DFEAS
- MSK_DPAR_INTPNT_CO_TOL_INFEAS
- MSK_DPAR_INTPNT_CO_TOL_MU_RED
- MSK_DPAR_INTPNT_CO_TOL_NEAR_REL
- MSK_DPAR_INTPNT_CO_TOL_PFEAS
- MSK_DPAR_INTPNT_CO_TOL_REL_GAP

Data check

- MSK_DPAR_DATA_SYM_MAT_TOL
- MSK_DPAR_DATA_SYM_MAT_TOL_HUGE
- MSK_DPAR_DATA_SYM_MAT_TOL_LARGE
- MSK_DPAR_DATA_TOL_AIJ_HUGE
- MSK_DPAR_DATA_TOL_AIJ_LARGE
- MSK_DPAR_DATA_TOL_BOUND_INF
- MSK_DPAR_DATA_TOL_BOUND_WRN
- MSK_DPAR_DATA_TOL_C_HUGE
- MSK_DPAR_DATA_TOL_CJ_LARGE
- MSK_DPAR_DATA_TOL_QIJ
- MSK_DPAR_DATA_TOL_X
- MSK_DPAR_SEMIDEFINITE_TOL_APPROX
- MSK_IPAR_CHECK_CONVEXITY
- MSK_IPAR_LOG_CHECK_CONVEXITY

Data input/output

- MSK_IPAR_INFEAS_REPORT_AUTO
- MSK_IPAR_LOG_FILE
- MSK_IPAR_OPF_WRITE_HEADER
- MSK_IPAR_OPF_WRITE_HINTS
- MSK_IPAR_OPF_WRITE_LINE_LENGTH
- MSK_IPAR_OPF_WRITE_PARAMETERS
- MSK_IPAR_OPF_WRITE_PROBLEM
- MSK_IPAR_OPF_WRITE_SOL_BAS
- MSK_IPAR_OPF_WRITE_SOL_ITG
- MSK_IPAR_OPF_WRITE_SOL_ITR
- MSK_IPAR_OPF_WRITE_SOLUTIONS

- *MSK_IPAR_PARAM_READ_CASE_NAME*
- *MSK_IPAR_PARAM_READ_IGN_ERROR*
- *MSK_IPAR_PTF_WRITE_TRANSFORM*
- *MSK_IPAR_READ_DEBUG*
- *MSK_IPAR_READ_KEEP_FREE_CON*
- *MSK_IPAR_READ_LP_DROP_NEW_VARS_IN_BOU*
- *MSK_IPAR_READ_LP_QUOTED_NAMES*
- *MSK_IPAR_READ_MPS_FORMAT*
- *MSK_IPAR_READ_MPS_WIDTH*
- *MSK_IPAR_READ_TASK_IGNORE_PARAM*
- *MSK_IPAR_SOL_READ_NAME_WIDTH*
- *MSK_IPAR_SOL_READ_WIDTH*
- *MSK_IPAR_WRITE_BAS_CONSTRAINTS*
- *MSK_IPAR_WRITE_BAS_HEAD*
- *MSK_IPAR_WRITE_BAS_VARIABLES*
- *MSK_IPAR_WRITE_COMPRESSION*
- *MSK_IPAR_WRITE_DATA_PARAM*
- *MSK_IPAR_WRITE_FREE_CON*
- *MSK_IPAR_WRITE_GENERIC_NAMES*
- *MSK_IPAR_WRITE_GENERIC_NAMES_IO*
- *MSK_IPAR_WRITE_IGNORE_INCOMPATIBLE_ITEMS*
- *MSK_IPAR_WRITE_INT_CONSTRAINTS*
- *MSK_IPAR_WRITE_INT_HEAD*
- *MSK_IPAR_WRITE_INT_VARIABLES*
- *MSK_IPAR_WRITE_LP_FULL_OBJ*
- *MSK_IPAR_WRITE_LP_LINE_WIDTH*
- *MSK_IPAR_WRITE_LP_QUOTED_NAMES*
- *MSK_IPAR_WRITE_LP_STRICT_FORMAT*
- *MSK_IPAR_WRITE_LP_TERMS_PER_LINE*
- *MSK_IPAR_WRITE_MPS_FORMAT*
- *MSK_IPAR_WRITE_MPS_INT*
- *MSK_IPAR_WRITE_PRECISION*
- *MSK_IPAR_WRITE_SOL_BARVARIABLES*
- *MSK_IPAR_WRITE_SOL_CONSTRAINTS*
- *MSK_IPAR_WRITE_SOL_HEAD*
- *MSK_IPAR_WRITE_SOL_IGNORE_INVALID_NAMES*

- *MSK_IPAR_WRITE_SOL_VARIABLES*
- *MSK_IPAR_WRITE_TASK_INC_SOL*
- *MSK_IPAR_WRITE_XML_MODE*
- *MSK_SPAR_BAS_SOL_FILE_NAME*
- *MSK_SPAR_DATA_FILE_NAME*
- *MSK_SPAR_DEBUG_FILE_NAME*
- *MSK_SPAR_INT_SOL_FILE_NAME*
- *MSK_SPAR_ITR_SOL_FILE_NAME*
- *MSK_SPAR_MIO_DEBUG_STRING*
- *MSK_SPAR_PARAM_COMMENT_SIGN*
- *MSK_SPAR_PARAM_READ_FILE_NAME*
- *MSK_SPAR_PARAM_WRITE_FILE_NAME*
- *MSK_SPAR_READ_MPS_BOU_NAME*
- *MSK_SPAR_READ_MPS_OBJ_NAME*
- *MSK_SPAR_READ_MPS_RAN_NAME*
- *MSK_SPAR_READ_MPS_RHS_NAME*
- *MSK_SPAR_SENSITIVITY_FILE_NAME*
- *MSK_SPAR_SENSITIVITY_RES_FILE_NAME*
- *MSK_SPAR_SOL_FILTER_XC_LOW*
- *MSK_SPAR_SOL_FILTER_XC_UPR*
- *MSK_SPAR_SOL_FILTER_XX_LOW*
- *MSK_SPAR_SOL_FILTER_XX_UPR*
- *MSK_SPAR_STAT_FILE_NAME*
- *MSK_SPAR_STAT_KEY*
- *MSK_SPAR_STAT_NAME*
- *MSK_SPAR_WRITE_LP_GEN_VAR_NAME*

Debugging

- *MSK_IPAR_AUTO_SORT_A_BEFORE_OPT*

Dual simplex

- *MSK_IPAR_SIM_DUAL_CRASH*
- *MSK_IPAR_SIM_DUAL_RESTRICT_SELECTION*
- *MSK_IPAR_SIM_DUAL_SELECTION*

Infeasibility report

- *MSK_IPAR_INFEAS_GENERIC_NAMES*
- *MSK_IPAR_INFEAS_REPORT_LEVEL*
- *MSK_IPAR_LOG_INFEAS_ANA*

Interior-point method

- *MSK_DPAR_CHECK_CONVEXITY_REL_TOL*
- *MSK_DPAR_INTPNT_CO_TOL_DFEAS*
- *MSK_DPAR_INTPNT_CO_TOL_INFEAS*
- *MSK_DPAR_INTPNT_CO_TOL_MU_RED*
- *MSK_DPAR_INTPNT_CO_TOL_NEAR_REL*
- *MSK_DPAR_INTPNT_CO_TOL_PFEAS*
- *MSK_DPAR_INTPNT_CO_TOL_REL_GAP*
- *MSK_DPAR_INTPNT_QO_TOL_DFEAS*
- *MSK_DPAR_INTPNT_QO_TOL_INFEAS*
- *MSK_DPAR_INTPNT_QO_TOL_MU_RED*
- *MSK_DPAR_INTPNT_QO_TOL_NEAR_REL*
- *MSK_DPAR_INTPNT_QO_TOL_PFEAS*
- *MSK_DPAR_INTPNT_QO_TOL_REL_GAP*
- *MSK_DPAR_INTPNT_TOL_DFEAS*
- *MSK_DPAR_INTPNT_TOL_DSAFE*
- *MSK_DPAR_INTPNT_TOL_INFEAS*
- *MSK_DPAR_INTPNT_TOL_MU_RED*
- *MSK_DPAR_INTPNT_TOL_PATH*
- *MSK_DPAR_INTPNT_TOL_PFEAS*
- *MSK_DPAR_INTPNT_TOL_PSAFE*
- *MSK_DPAR_INTPNT_TOL_REL_GAP*
- *MSK_DPAR_INTPNT_TOL_REL_STEP*
- *MSK_DPAR_INTPNT_TOL_STEP_SIZE*
- *MSK_DPAR_QCQO_REFORMULATE_REL_DROP_TOL*
- *MSK_IPAR_BI_IGNORE_MAX_ITER*
- *MSK_IPAR_BI_IGNORE_NUM_ERROR*
- *MSK_IPAR_INTPNT_BASIS*
- *MSK_IPAR_INTPNT_DIFF_STEP*
- *MSK_IPAR_INTPNT_HOTSTART*
- *MSK_IPAR_INTPNT_MAX_ITERATIONS*

- *MSK_IPAR_INTPNT_MAX_NUM_COR*
- *MSK_IPAR_INTPNT_MAX_NUM_REFINEMENT_STEPS*
- *MSK_IPAR_INTPNT_OFF_COL_TRH*
- *MSK_IPAR_INTPNT_ORDER_METHOD*
- *MSK_IPAR_INTPNT_PURIFY*
- *MSK_IPAR_INTPNT_REGULARIZATION_USE*
- *MSK_IPAR_INTPNT_SCALING*
- *MSK_IPAR_INTPNT_SOLVE_FORM*
- *MSK_IPAR_INTPNT_STARTING_POINT*
- *MSK_IPAR_LOG_INTPNT*

License manager

- *MSK_IPAR_CACHE_LICENSE*
- *MSK_IPAR_LICENSE_DEBUG*
- *MSK_IPAR_LICENSE_PAUSE_TIME*
- *MSK_IPAR_LICENSE_SUPPRESS_EXPIRE_WRNS*
- *MSK_IPAR_LICENSE_TRH_EXPIRY_WRN*
- *MSK_IPAR_LICENSE_WAIT*

Logging

- *MSK_IPAR_LOG*
- *MSK_IPAR_LOG_ANA_PRO*
- *MSK_IPAR_LOG_BI*
- *MSK_IPAR_LOG_BI_FREQ*
- *MSK_IPAR_LOG_CUT_SECOND_OPT*
- *MSK_IPAR_LOG_EXPAND*
- *MSK_IPAR_LOG_FEAS_REPAIR*
- *MSK_IPAR_LOG_FILE*
- *MSK_IPAR_LOG_INFEAS_ANA*
- *MSK_IPAR_LOG_INTPNT*
- *MSK_IPAR_LOG_LOCAL_INFO*
- *MSK_IPAR_LOG_MIO*
- *MSK_IPAR_LOG_MIO_FREQ*
- *MSK_IPAR_LOG_ORDER*
- *MSK_IPAR_LOG_PRESOLVE*
- *MSK_IPAR_LOG_RESPONSE*

- *MSK_IPAR_LOG_SENSITIVITY*
- *MSK_IPAR_LOG_SENSITIVITY_OPT*
- *MSK_IPAR_LOG_SIM*
- *MSK_IPAR_LOG_SIM_FREQ*
- *MSK_IPAR_LOG_STORAGE*

Mixed-integer optimization

- *MSK_DPAR_MIO_MAX_TIME*
- *MSK_DPAR_MIO_REL_GAP_CONST*
- *MSK_DPAR_MIO_TOL_ABS_GAP*
- *MSK_DPAR_MIO_TOL_ABS_RELAX_INT*
- *MSK_DPAR_MIO_TOL_FEAS*
- *MSK_DPAR_MIO_TOL_REL_DUAL_BOUND_IMPROVEMENT*
- *MSK_DPAR_MIO_TOL_REL_GAP*
- *MSK_IPAR_LOG_MIO*
- *MSK_IPAR_LOG_MIO_FREQ*
- *MSK_IPAR_MIO_BRANCH_DIR*
- *MSK_IPAR_MIO_CONIC_OUTER_APPROXIMATION*
- *MSK_IPAR_MIO_CUT_CLIQUE*
- *MSK_IPAR_MIO_CUT_CMIR*
- *MSK_IPAR_MIO_CUT_GMI*
- *MSK_IPAR_MIO_CUT_IMPLIED_BOUND*
- *MSK_IPAR_MIO_CUT_KNAPSACK_COVER*
- *MSK_IPAR_MIO_CUT_SELECTION_LEVEL*
- *MSK_IPAR_MIO_FEASPUMP_LEVEL*
- *MSK_IPAR_MIO_HEURISTIC_LEVEL*
- *MSK_IPAR_MIO_MAX_NUM_BRANCHES*
- *MSK_IPAR_MIO_MAX_NUM_RELAXS*
- *MSK_IPAR_MIO_MAX_NUM_ROOT_CUT_ROUNDS*
- *MSK_IPAR_MIO_MAX_NUM_SOLUTIONS*
- *MSK_IPAR_MIO_NODE_OPTIMIZER*
- *MSK_IPAR_MIO_NODE_SELECTION*
- *MSK_IPAR_MIO_PERSPECTIVE_REFORMULATE*
- *MSK_IPAR_MIO_PROBING_LEVEL*
- *MSK_IPAR_MIO_PROPAGATE_OBJECTIVE_CONSTRAINT*
- *MSK_IPAR_MIO_RINS_MAX_NODES*

- *MSK_IPAR_MIO_ROOT_OPTIMIZER*
- *MSK_IPAR_MIO_ROOT_REPEAT_PRESOLVE_LEVEL*
- *MSK_IPAR_MIO_SEED*
- *MSK_IPAR_MIO_VB_DETECTION_LEVEL*

Output information

- *MSK_IPAR_INFEAS_REPORT_LEVEL*
- *MSK_IPAR_LICENSE_SUPPRESS_EXPIRE_WRNS*
- *MSK_IPAR_LICENSE_TRH_EXPIRY_WRN*
- *MSK_IPAR_LOG*
- *MSK_IPAR_LOG_BI*
- *MSK_IPAR_LOG_BI_FREQ*
- *MSK_IPAR_LOG_CUT_SECOND_OPT*
- *MSK_IPAR_LOG_EXPAND*
- *MSK_IPAR_LOG_FEAS_REPAIR*
- *MSK_IPAR_LOG_FILE*
- *MSK_IPAR_LOG_INFEAS_ANA*
- *MSK_IPAR_LOG_INTPNT*
- *MSK_IPAR_LOG_LOCAL_INFO*
- *MSK_IPAR_LOG_MIO*
- *MSK_IPAR_LOG_MIO_FREQ*
- *MSK_IPAR_LOG_ORDER*
- *MSK_IPAR_LOG_RESPONSE*
- *MSK_IPAR_LOG_SENSITIVITY*
- *MSK_IPAR_LOG_SENSITIVITY_OPT*
- *MSK_IPAR_LOG_SIM*
- *MSK_IPAR_LOG_SIM_FREQ*
- *MSK_IPAR_LOG_SIM_MINOR*
- *MSK_IPAR_LOG_STORAGE*
- *MSK_IPAR_MAX_NUM_WARNINGS*

Overall solver

- *MSK_IPAR_BI_CLEAN_OPTIMIZER*
- *MSK_IPAR_INFEAS_PREFER_PRIMAL*
- *MSK_IPAR_LICENSE_WAIT*
- *MSK_IPAR_MIO_MODE*
- *MSK_IPAR_OPTIMIZER*
- *MSK_IPAR_PRESOLVE_LEVEL*
- *MSK_IPAR_PRESOLVE_MAX_NUM_REDUCTIONS*
- *MSK_IPAR_PRESOLVE_USE*
- *MSK_IPAR_PRIMAL_REPAIR_OPTIMIZER*
- *MSK_IPAR_SENSITIVITY_ALL*
- *MSK_IPAR_SENSITIVITY_OPTIMIZER*
- *MSK_IPAR_SENSITIVITY_TYPE*
- *MSK_IPAR_SOLUTION_CALLBACK*

Overall system

- *MSK_IPAR_AUTO_UPDATE_SOL_INFO*
- *MSK_IPAR_INTPNT_MULTI_THREAD*
- *MSK_IPAR_LICENSE_WAIT*
- *MSK_IPAR_LOG_STORAGE*
- *MSK_IPAR_MT_SPINCOUNT*
- *MSK_IPAR_NUM_THREADS*
- *MSK_IPAR_REMOVE_UNUSED_SOLUTIONS*
- *MSK_IPAR_TIMING_LEVEL*
- *MSK_SPAR_REMOTE_ACCESS_TOKEN*

Presolve

- *MSK_DPAR_PRESOLVE_TOL_ABS_LINDEP*
- *MSK_DPAR_PRESOLVE_TOL_AIJ*
- *MSK_DPAR_PRESOLVE_TOL_REL_LINDEP*
- *MSK_DPAR_PRESOLVE_TOL_S*
- *MSK_DPAR_PRESOLVE_TOL_X*
- *MSK_IPAR_PRESOLVE_ELIMINATOR_MAX_FILL*
- *MSK_IPAR_PRESOLVE_ELIMINATOR_MAX_NUM_TRIES*
- *MSK_IPAR_PRESOLVE_LEVEL*
- *MSK_IPAR_PRESOLVE_LINDEP_ABS_WORK_TRH*

- *MSK_IPAR_PRESOLVE_LINDEP_REL_WORK_TRH*
- *MSK_IPAR_PRESOLVE_LINDEP_USE*
- *MSK_IPAR_PRESOLVE_MAX_NUM_PASS*
- *MSK_IPAR_PRESOLVE_MAX_NUM_REDUCTIONS*
- *MSK_IPAR_PRESOLVE_USE*

Primal simplex

- *MSK_IPAR_SIM_PRIMAL_CRASH*
- *MSK_IPAR_SIM_PRIMAL_RESTRICT_SELECTION*
- *MSK_IPAR_SIM_PRIMAL_SELECTION*

Progress callback

- *MSK_IPAR_SOLUTION_CALLBACK*

Simplex optimizer

- *MSK_DPAR_BASIS_REL_TOL_S*
- *MSK_DPAR_BASIS_TOL_S*
- *MSK_DPAR_BASIS_TOL_X*
- *MSK_DPAR_SIM_LU_TOL_REL_PIV*
- *MSK_DPAR_SIMPLEX_ABS_TOL_PIV*
- *MSK_IPAR_BASIS_SOLVE_USE_PLUS_ONE*
- *MSK_IPAR_LOG_SIM*
- *MSK_IPAR_LOG_SIM_FREQ*
- *MSK_IPAR_LOG_SIM_MINOR*
- *MSK_IPAR_SENSITIVITY_OPTIMIZER*
- *MSK_IPAR_SIM_BASIS_FACTOR_USE*
- *MSK_IPAR_SIM_DEGEN*
- *MSK_IPAR_SIM_DUAL_PHASEONE_METHOD*
- *MSK_IPAR_SIM_EXPLOIT_DUPVEC*
- *MSK_IPAR_SIM_HOTSTART*
- *MSK_IPAR_SIM_HOTSTART_LU*
- *MSK_IPAR_SIM_MAX_ITERATIONS*
- *MSK_IPAR_SIM_MAX_NUM_SETBACKS*
- *MSK_IPAR_SIM_NON_SINGULAR*
- *MSK_IPAR_SIM_PRIMAL_PHASEONE_METHOD*
- *MSK_IPAR_SIM_REFACTOR_FREQ*

- *MSK_IPAR_SIM_REFORMULATION*
- *MSK_IPAR_SIM_SAVE_LU*
- *MSK_IPAR_SIM_SCALING*
- *MSK_IPAR_SIM_SCALING_METHOD*
- *MSK_IPAR_SIM_SEED*
- *MSK_IPAR_SIM_SOLVE_FORM*
- *MSK_IPAR_SIM_STABILITY_PRIORITY*
- *MSK_IPAR_SIM_SWITCH_OPTIMIZER*

Solution input/output

- *MSK_IPAR_INFEAS_REPORT_AUTO*
- *MSK_IPAR_SOL_FILTER_KEEP_BASIC*
- *MSK_IPAR_SOL_FILTER_KEEP_RANGED*
- *MSK_IPAR_SOL_READ_NAME_WIDTH*
- *MSK_IPAR_SOL_READ_WIDTH*
- *MSK_IPAR_WRITE_BAS_CONSTRAINTS*
- *MSK_IPAR_WRITE_BAS_HEAD*
- *MSK_IPAR_WRITE_BAS_VARIABLES*
- *MSK_IPAR_WRITE_INT_CONSTRAINTS*
- *MSK_IPAR_WRITE_INT_HEAD*
- *MSK_IPAR_WRITE_INT_VARIABLES*
- *MSK_IPAR_WRITE_SOL_BARVARIABLES*
- *MSK_IPAR_WRITE_SOL_CONSTRAINTS*
- *MSK_IPAR_WRITE_SOL_HEAD*
- *MSK_IPAR_WRITE_SOL_IGNORE_INVALID_NAMES*
- *MSK_IPAR_WRITE_SOL_VARIABLES*
- *MSK_SPAR_BAS_SOL_FILE_NAME*
- *MSK_SPAR_INT_SOL_FILE_NAME*
- *MSK_SPAR_ITR_SOL_FILE_NAME*
- *MSK_SPAR_SOL_FILTER_XC_LOW*
- *MSK_SPAR_SOL_FILTER_XC_UPR*
- *MSK_SPAR_SOL_FILTER_XX_LOW*
- *MSK_SPAR_SOL_FILTER_XX_UPR*

Termination criteria

- *MSK_DPAR_BASIS_REL_TOL_S*
- *MSK_DPAR_BASIS_TOL_S*
- *MSK_DPAR_BASIS_TOL_X*
- *MSK_DPAR_INTPNT_CO_TOL_DFEAS*
- *MSK_DPAR_INTPNT_CO_TOL_INFEAS*
- *MSK_DPAR_INTPNT_CO_TOL_MU_RED*
- *MSK_DPAR_INTPNT_CO_TOL_NEAR_REL*
- *MSK_DPAR_INTPNT_CO_TOL_PFEAS*
- *MSK_DPAR_INTPNT_CO_TOL_REL_GAP*
- *MSK_DPAR_INTPNT_QO_TOL_DFEAS*
- *MSK_DPAR_INTPNT_QO_TOL_INFEAS*
- *MSK_DPAR_INTPNT_QO_TOL_MU_RED*
- *MSK_DPAR_INTPNT_QO_TOL_NEAR_REL*
- *MSK_DPAR_INTPNT_QO_TOL_PFEAS*
- *MSK_DPAR_INTPNT_QO_TOL_REL_GAP*
- *MSK_DPAR_INTPNT_TOL_DFEAS*
- *MSK_DPAR_INTPNT_TOL_INFEAS*
- *MSK_DPAR_INTPNT_TOL_MU_RED*
- *MSK_DPAR_INTPNT_TOL_PFEAS*
- *MSK_DPAR_INTPNT_TOL_REL_GAP*
- *MSK_DPAR_LOWER_OBJ_CUT*
- *MSK_DPAR_LOWER_OBJ_CUT_FINITE_TRH*
- *MSK_DPAR_MIO_MAX_TIME*
- *MSK_DPAR_MIO_REL_GAP_CONST*
- *MSK_DPAR_MIO_TOL_REL_GAP*
- *MSK_DPAR_OPTIMIZER_MAX_TIME*
- *MSK_DPAR_UPPER_OBJ_CUT*
- *MSK_DPAR_UPPER_OBJ_CUT_FINITE_TRH*
- *MSK_IPAR_BI_MAX_ITERATIONS*
- *MSK_IPAR_INTPNT_MAX_ITERATIONS*
- *MSK_IPAR_MIO_MAX_NUM_BRANCHES*
- *MSK_IPAR_MIO_MAX_NUM_ROOT_CUT_ROUNDS*
- *MSK_IPAR_MIO_MAX_NUM_SOLUTIONS*
- *MSK_IPAR_SIM_MAX_ITERATIONS*

Other

- *MSK_IPAR_COMPRESS_STATFILE*

15.5 Parameters (alphabetical list sorted by type)

- *Double parameters*
- *Integer parameters*
- *String parameters*

15.5.1 Double parameters

`dparam`
The enumeration type containing all double parameters.

`MSK_DPAR_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*
Example `param.MSK_DPAR_ANA_SOL_INFEAS_TOL = 1e-6`

`MSK_DPAR_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*
Example `param.MSK_DPAR_BASIS_REL_TOL_S = 1.0e-12`

`MSK_DPAR_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*
Example `param.MSK_DPAR_BASIS_TOL_S = 1.0e-6`

`MSK_DPAR_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*
Example `param.MSK_DPAR_BASIS_TOL_X = 1.0e-6`

`MSK_DPAR_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 nonnegativity requirement may be violated.

If d_i is the pivot element for column i, then the matrix Q is considered to not be PSD if:

$$d_i \leq -|Q_{ii}|\texttt{check\_convexity\_rel\_tol}$$

Default 1e-10

Accepted [0; +inf]

Groups *Interior-point method*

Example `param.MSK_DPAR_CHECK_CONVEXITY_REL_TOL = 1e-10`

`MSK_DPAR_DATA_SYM_MAT_TOL`

Absolute zero tolerance for elements in in symmetric matrices. If any value in a symmetric matrix is smaller than this parameter in absolute terms **MOSEK** will treat the values as zero and generate a warning.

Default 1.0e-12

Accepted [1.0e-16; 1.0e-6]

Groups *Data check*

Example `param.MSK_DPAR_DATA_SYM_MAT_TOL = 1.0e-12`

`MSK_DPAR_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*

Example `param.MSK_DPAR_DATA_SYM_MAT_TOL_HUGE = 1.0e20`

`MSK_DPAR_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*

Example `param.MSK_DPAR_DATA_SYM_MAT_TOL_LARGE = 1.0e10`

`MSK_DPAR_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*

Example `param.MSK_DPAR_DATA_TOL_AIJ_HUGE = 1.0e20`

`MSK_DPAR_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*

Example `param.MSK_DPAR_DATA_TOL_AIJ_LARGE = 1.0e10`

`MSK_DPAR_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*

Example `param.MSK_DPAR_DATA_TOL_BOUND_INF = 1.0e16`

`MSK_DPAR_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*

Example `param.MSK_DPAR_DATA_TOL_BOUND_WRN = 1.0e8`

`MSK_DPAR_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*

Example `param.MSK_DPAR_DATA_TOL_C_HUGE = 1.0e16`

`MSK_DPAR_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*

Example `param.MSK_DPAR_DATA_TOL_CJ_LARGE = 1.0e8`

`MSK_DPAR_DATA_TOL_QIJ`

Absolute zero tolerance for elements in Q matrices.

Default 1.0e-16

Accepted [0.0; +inf]

Groups *Data check*

Example `param.MSK_DPAR_DATA_TOL_QIJ = 1.0e-16`

`MSK_DPAR_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*

Example `param.MSK_DPAR_DATA_TOL_X = 1.0e-8`

`MSK_DPAR_INTPNT_CO_TOL_DFEAS`

Dual feasibility tolerance used by the interior-point optimizer for conic problems.

Default 1.0e-8

Accepted [0.0; 1.0]

Groups *Interior-point method*, *Termination criteria*, *Conic interior-point method*

See also *MSK_DPAR_INTPNT_CO_TOL_NEAR_REL*

Example `param.MSK_DPAR_INTPNT_CO_TOL_DFEAS = 1.0e-8`

`MSK_DPAR_INTPNT_CO_TOL_INFEAS`

Infeasibility tolerance used by the interior-point optimizer for conic problems. Controls when the interior-point optimizer declares the model primal or dual infeasible. A small number means the optimizer gets more conservative about declaring the model infeasible.

Default 1.0e-12

Accepted [0.0; 1.0]

Groups *Interior-point method*, *Termination criteria*, *Conic interior-point method*

Example `param.MSK_DPAR_INTPNT_CO_TOL_INFEAS = 1.0e-12`

MSK_DPAR_INTPNT_CO_TOL_MU_RED
Relative complementarity gap tolerance used by the interior-point optimizer for conic problems.

Default 1.0e-8
Accepted [0.0; 1.0]
Groups *Interior-point method*, *Termination criteria*, *Conic interior-point method*
Example `param.MSK_DPAR_INTPNT_CO_TOL_MU_RED = 1.0e-8`

MSK_DPAR_INTPNT_CO_TOL_NEAR_REL
Optimality tolerance used by the interior-point optimizer for conic problems. If **MOSEK** cannot compute a solution that has the prescribed accuracy then it will check if the solution found satisfies the termination criteria with all tolerances multiplied by the value of this parameter. If yes, then the solution is also declared optimal.

Default 1000
Accepted [1.0; +inf]
Groups *Interior-point method*, *Termination criteria*, *Conic interior-point method*
Example `param.MSK_DPAR_INTPNT_CO_TOL_NEAR_REL = 1000`

MSK_DPAR_INTPNT_CO_TOL_PFEAS
Primal feasibility tolerance used by the interior-point optimizer for conic problems.

Default 1.0e-8
Accepted [0.0; 1.0]
Groups *Interior-point method*, *Termination criteria*, *Conic interior-point method*
See also *MSK_DPAR_INTPNT_CO_TOL_NEAR_REL*
Example `param.MSK_DPAR_INTPNT_CO_TOL_PFEAS = 1.0e-8`

MSK_DPAR_INTPNT_CO_TOL_REL_GAP
Relative gap termination tolerance used by the interior-point optimizer for conic problems.

Default 1.0e-8
Accepted [0.0; 1.0]
Groups *Interior-point method*, *Termination criteria*, *Conic interior-point method*
See also *MSK_DPAR_INTPNT_CO_TOL_NEAR_REL*
Example `param.MSK_DPAR_INTPNT_CO_TOL_REL_GAP = 1.0e-8`

MSK_DPAR_INTPNT_QO_TOL_DFEAS
Dual feasibility tolerance used by the interior-point optimizer for quadratic problems.

Default 1.0e-8
Accepted [0.0; 1.0]
Groups *Interior-point method*, *Termination criteria*
See also *MSK_DPAR_INTPNT_QO_TOL_NEAR_REL*
Example `param.MSK_DPAR_INTPNT_QO_TOL_DFEAS = 1.0e-8`

MSK_DPAR_INTPNT_QO_TOL_INFEAS
Infeasibility tolerance used by the interior-point optimizer for quadratic problems. Controls when the interior-point optimizer declares the model primal or dual infeasible. A small number means the optimizer gets more conservative about declaring the model infeasible.

Default 1.0e-12
Accepted [0.0; 1.0]
Groups *Interior-point method*, *Termination criteria*
Example `param.MSK_DPAR_INTPNT_QO_TOL_INFEAS = 1.0e-12`

MSK_DPAR_INTPNT_QO_TOL_MU_RED
Relative complementarity gap tolerance used by the interior-point optimizer for quadratic problems.

Default 1.0e-8

Accepted [0.0; 1.0]

Groups *Interior-point method*, *Termination criteria*

Example `param.MSK_DPAR_INTPNT_QO_TOL_MU_RED = 1.0e-8`

`MSK_DPAR_INTPNT_QO_TOL_NEAR_REL`

Optimality tolerance used by the interior-point optimizer for quadratic problems. If **MOSEK** cannot compute a solution that has the prescribed accuracy then it will check if the solution found satisfies the termination criteria with all tolerances multiplied by the value of this parameter. If yes, then the solution is also declared optimal.

Default 1000

Accepted [1.0; +inf]

Groups *Interior-point method*, *Termination criteria*

Example `param.MSK_DPAR_INTPNT_QO_TOL_NEAR_REL = 1000`

`MSK_DPAR_INTPNT_QO_TOL_PFEAS`

Primal feasibility tolerance used by the interior-point optimizer for quadratic problems.

Default 1.0e-8

Accepted [0.0; 1.0]

Groups *Interior-point method*, *Termination criteria*

See also *MSK_DPAR_INTPNT_QO_TOL_NEAR_REL*

Example `param.MSK_DPAR_INTPNT_QO_TOL_PFEAS = 1.0e-8`

`MSK_DPAR_INTPNT_QO_TOL_REL_GAP`

Relative gap termination tolerance used by the interior-point optimizer for quadratic problems.

Default 1.0e-8

Accepted [0.0; 1.0]

Groups *Interior-point method*, *Termination criteria*

See also *MSK_DPAR_INTPNT_QO_TOL_NEAR_REL*

Example `param.MSK_DPAR_INTPNT_QO_TOL_REL_GAP = 1.0e-8`

`MSK_DPAR_INTPNT_TOL_DFEAS`

Dual feasibility tolerance used by the interior-point optimizer for linear problems.

Default 1.0e-8

Accepted [0.0; 1.0]

Groups *Interior-point method*, *Termination criteria*

Example `param.MSK_DPAR_INTPNT_TOL_DFEAS = 1.0e-8`

`MSK_DPAR_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*

Example `param.MSK_DPAR_INTPNT_TOL_DSAFE = 1.0`

`MSK_DPAR_INTPNT_TOL_INFEAS`

Infeasibility tolerance used by the interior-point optimizer for linear problems. Controls when the interior-point optimizer declares the model primal or dual infeasible. A small number means the optimizer gets more conservative about declaring the model infeasible.

Default 1.0e-10

Accepted [0.0; 1.0]

Groups *Interior-point method*, *Termination criteria*

Example `param.MSK_DPAR_INTPNT_TOL_INFEAS = 1.0e-10`

`MSK_DPAR_INTPNT_TOL_MU_RED`

Relative complementarity gap tolerance used by the interior-point optimizer for linear problems.

Default 1.0e-16

Accepted [0.0; 1.0]

Groups *Interior-point method*, *Termination criteria*

Example `param.MSK_DPAR_INTPNT_TOL_MU_RED = 1.0e-16`

`MSK_DPAR_INTPNT_TOL_PATH`

Controls how close the interior-point optimizer follows the central path. A large value of this parameter means the central path is followed very closely. On numerically unstable problems it may be worthwhile to increase this parameter.

Default 1.0e-8

Accepted [0.0; 0.9999]

Groups *Interior-point method*

Example `param.MSK_DPAR_INTPNT_TOL_PATH = 1.0e-8`

`MSK_DPAR_INTPNT_TOL_PFEAS`

Primal feasibility tolerance used by the interior-point optimizer for linear problems.

Default 1.0e-8

Accepted [0.0; 1.0]

Groups *Interior-point method*, *Termination criteria*

Example `param.MSK_DPAR_INTPNT_TOL_PFEAS = 1.0e-8`

`MSK_DPAR_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*

Example `param.MSK_DPAR_INTPNT_TOL_PSAFE = 1.0`

`MSK_DPAR_INTPNT_TOL_REL_GAP`

Relative gap termination tolerance used by the interior-point optimizer for linear problems.

Default 1.0e-8

Accepted [1.0e-14; +inf]

Groups *Termination criteria*, *Interior-point method*

Example `param.MSK_DPAR_INTPNT_TOL_REL_GAP = 1.0e-8`

`MSK_DPAR_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*

Example `param.MSK_DPAR_INTPNT_TOL_REL_STEP = 0.9999`

`MSK_DPAR_INTPNT_TOL_STEP_SIZE`

Minimal step size tolerance. If the step size falls below the value of this parameter, then the interior-point optimizer assumes that it is stalled. In other words the interior-point optimizer does not make any progress and therefore it is better to stop.

Default 1.0e-6

Accepted [0.0; 1.0]

Groups *Interior-point method*

Example `param.MSK_DPAR_INTPNT_TOL_STEP_SIZE = 1.0e-6`

`MSK_DPAR_LOWER_OBJ_CUT`

If either a primal or dual feasible solution is found proving that the optimal objective value is outside the interval [*MSK_DPAR_LOWER_OBJ_CUT*, *MSK_DPAR_UPPER_OBJ_CUT*], then **MOSEK** is terminated.

Default -1.0e30

Accepted [-inf; +inf]

Groups *Termination criteria*

See also *MSK_DPAR_LOWER_OBJ_CUT_FINITE_TRH*

Example `param.MSK_DPAR_LOWER_OBJ_CUT = -1.0e30`

`MSK_DPAR_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. *MSK_DPAR_LOWER_OBJ_CUT* is treated as $-\infty$.

Default -0.5e30

Accepted [-inf; +inf]

Groups *Termination criteria*

Example `param.MSK_DPAR_LOWER_OBJ_CUT_FINITE_TRH = -0.5e30`

`MSK_DPAR_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*

Example `param.MSK_DPAR_MIO_MAX_TIME = -1.0`

`MSK_DPAR_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*

Example `param.MSK_DPAR_MIO_REL_GAP_CONST = 1.0e-10`

`MSK_DPAR_MIO_TOL_ABS_GAP`

Absolute optimality tolerance employed by the mixed-integer optimizer.

Default 0.0

Accepted [0.0; +inf]

Groups *Mixed-integer optimization*

Example `param.MSK_DPAR_MIO_TOL_ABS_GAP = 0.0`

`MSK_DPAR_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*

Example `param.MSK_DPAR_MIO_TOL_ABS_RELAX_INT = 1.0e-5`

`MSK_DPAR_MIO_TOL_FEAS`

Feasibility tolerance for mixed integer solver.

Default 1.0e-6
Accepted [1e-9; 1e-3]
Groups *Mixed-integer optimization*
Example `param.MSK_DPAR_MIO_TOL_FEAS = 1.0e-6`

`MSK_DPAR_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*
Example `param.MSK_DPAR_MIO_TOL_REL_DUAL_BOUND_IMPROVEMENT = 0.0`

`MSK_DPAR_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*
Example `param.MSK_DPAR_MIO_TOL_REL_GAP = 1.0e-4`

`MSK_DPAR_OPTIMIZER_MAX_TIME`

Maximum amount of time the optimizer is allowed to spent on the optimization. A negative number means infinity.

Default -1.0
Accepted [-inf; +inf]
Groups *Termination criteria*
Example `param.MSK_DPAR_OPTIMIZER_MAX_TIME = -1.0`

`MSK_DPAR_PRESOLVE_TOL_ABS_LINDEP`

Absolute tolerance employed by the linear dependency checker.

Default 1.0e-6
Accepted [0.0; +inf]
Groups *Presolve*
Example `param.MSK_DPAR_PRESOLVE_TOL_ABS_LINDEP = 1.0e-6`

`MSK_DPAR_PRESOLVE_TOL_AIJ`

Absolute zero tolerance employed for a_{ij} in the presolve.

Default 1.0e-12
Accepted [1.0e-15; +inf]
Groups *Presolve*
Example `param.MSK_DPAR_PRESOLVE_TOL_AIJ = 1.0e-12`

`MSK_DPAR_PRESOLVE_TOL_REL_LINDEP`

Relative tolerance employed by the linear dependency checker.

Default 1.0e-10
Accepted [0.0; +inf]
Groups *Presolve*
Example `param.MSK_DPAR_PRESOLVE_TOL_REL_LINDEP = 1.0e-10`

`MSK_DPAR_PRESOLVE_TOL_S`

Absolute zero tolerance employed for s_i in the presolve.

Default 1.0e-8
Accepted [0.0; +inf]

Groups *Presolve*

Example `param.MSK_DPAR_PRESOLVE_TOL_S = 1.0e-8`

`MSK_DPAR_PRESOLVE_TOL_X`

Absolute zero tolerance employed for x_j in the presolve.

Default 1.0e-8

Accepted [0.0; +inf]

Groups *Presolve*

Example `param.MSK_DPAR_PRESOLVE_TOL_X = 1.0e-8`

`MSK_DPAR_QCQO_REFORMULATE_REL_DROP_TOL`

This parameter determines when columns are dropped in incomplete Cholesky factorization during reformulation of quadratic problems.

Default 1e-15

Accepted [0; +inf]

Groups *Interior-point method*

Example `param.MSK_DPAR_QCQO_REFORMULATE_REL_DROP_TOL = 1e-15`

`MSK_DPAR_SEMIDEFINITE_TOL_APPROX`

Tolerance to define a matrix to be positive semidefinite.

Default 1.0e-10

Accepted [1.0e-15; +inf]

Groups *Data check*

Example `param.MSK_DPAR_SEMIDEFINITE_TOL_APPROX = 1.0e-10`

`MSK_DPAR_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*

Example `param.MSK_DPAR_SIM_LU_TOL_REL_PIV = 0.01`

`MSK_DPAR_SIMPLEX_ABS_TOL_PIV`

Absolute pivot tolerance employed by the simplex optimizers.

Default 1.0e-7

Accepted [1.0e-12; +inf]

Groups *Simplex optimizer*

Example `param.MSK_DPAR_SIMPLEX_ABS_TOL_PIV = 1.0e-7`

`MSK_DPAR_UPPER_OBJ_CUT`

If either a primal or dual feasible solution is found proving that the optimal objective value is outside the interval [*MSK_DPAR_LOWER_OBJ_CUT*, *MSK_DPAR_UPPER_OBJ_CUT*], then **MOSEK** is terminated.

Default 1.0e30

Accepted [-inf; +inf]

Groups *Termination criteria*

See also *MSK_DPAR_UPPER_OBJ_CUT_FINITE_TRH*

Example `param.MSK_DPAR_UPPER_OBJ_CUT = 1.0e30`

`MSK_DPAR_UPPER_OBJ_CUT_FINITE_TRH`

If the upper objective cut is greater than the value of this parameter, then the upper objective cut *MSK_DPAR_UPPER_OBJ_CUT* is treated as ∞.

Default 0.5e30

Accepted [-inf; +inf]

Groups *Termination criteria*

Example `param.MSK_DPAR_UPPER_OBJ_CUT_FINITE_TRH = 0.5e30`

15.5.2 Integer parameters

`iparam`

The enumeration type containing all integer parameters.

`MSK_IPAR_ANA_SOL_BASIS`

Controls whether the basis matrix is analyzed in solution analyzer.

Default *"ON"*

Accepted *"ON"*, *"OFF"*

Groups *Analysis*

Example `param.MSK_IPAR_ANA_SOL_BASIS = 'MSK_ON'`

`MSK_IPAR_ANA_SOL_PRINT_VIOLATED`

A parameter of the problem analyzer. Controls whether a list of violated constraints is printed. All constraints violated by more than the value set by the parameter *MSK_DPAR_ANA_SOL_INFEAS_TOL* will be printed.

Default *"OFF"*

Accepted *"ON"*, *"OFF"*

Groups *Analysis*

Example `param.MSK_IPAR_ANA_SOL_PRINT_VIOLATED = 'MSK_OFF'`

`MSK_IPAR_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"*

Groups *Debugging*

Example `param.MSK_IPAR_AUTO_SORT_A_BEFORE_OPT = 'MSK_OFF'`

`MSK_IPAR_AUTO_UPDATE_SOL_INFO`

Controls whether the solution information items are automatically updated after an optimization is performed.

Default *"OFF"*

Accepted *"ON"*, *"OFF"*

Groups *Overall system*

Example `param.MSK_IPAR_AUTO_UPDATE_SOL_INFO = 'MSK_OFF'`

`MSK_IPAR_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 *"MSK_ON"*, -1 is replaced by 1.

Default *"OFF"*

Accepted *"ON"*, *"OFF"*

Groups *Simplex optimizer*

Example `param.MSK_IPAR_BASIS_SOLVE_USE_PLUS_ONE = 'MSK_OFF'`

`MSK_IPAR_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"*

Groups *Basis identification*, *Overall solver*

Example `param.MSK_IPAR_BI_CLEAN_OPTIMIZER = 'MSK_OPTIMIZER_FREE'`

`MSK_IPAR_BI_IGNORE_MAX_ITER`

If the parameter *MSK_IPAR_INTPNT_BASIS* has the value *"MSK_BI_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 *"MSK_ON"*.

Default *"OFF"*

Accepted *"ON"*, *"OFF"*

Groups *Interior-point method*, *Basis identification*

Example `param.MSK_IPAR_BI_IGNORE_MAX_ITER = 'MSK_OFF'`

`MSK_IPAR_BI_IGNORE_NUM_ERROR`

If the parameter *MSK_IPAR_INTPNT_BASIS* has the value *"MSK_BI_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 *"MSK_ON"*.

Default *"OFF"*

Accepted *"ON"*, *"OFF"*

Groups *Interior-point method*, *Basis identification*

Example `param.MSK_IPAR_BI_IGNORE_NUM_ERROR = 'MSK_OFF'`

`MSK_IPAR_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*

Example `param.MSK_IPAR_BI_MAX_ITERATIONS = 1000000`

`MSK_IPAR_CACHE_LICENSE`

Specifies if the license is kept checked out for the lifetime of the **MOSEK** environment/model/process (*"MSK_ON"*) or returned to the server immediately after the optimization (*"MSK_OFF"*).

By default the license is checked out for the lifetime of the session at the start of first optimization.

Check-in and check-out of licenses have an overhead. Frequent communication with the license server should be avoided.

Default *"ON"*

Accepted *"ON"*, *"OFF"*

Groups *License manager*

Example `param.MSK_IPAR_CACHE_LICENSE = 'MSK_ON'`

`MSK_IPAR_CHECK_CONVEXITY`

Specify the level of convexity check on quadratic problems.

Default *"FULL"*

Accepted *"NONE"*, *"SIMPLE"*, *"FULL"*

Groups *Data check*

Example `param.MSK_IPAR_CHECK_CONVEXITY = 'MSK_CHECK_CONVEXITY_FULL'`

`MSK_IPAR_COMPRESS_STATFILE`

Control compression of stat files.

Default *"ON"*

Accepted *"ON"*, *"OFF"*

Example `param.MSK_IPAR_COMPRESS_STATFILE = 'MSK_ON'`

MSK_IPAR_INFEAS_GENERIC_NAMES

Controls whether generic names are used when an infeasible subproblem is created.

Default *"OFF"*

Accepted *"ON"*, *"OFF"*

Groups *Infeasibility report*

Example `param.MSK_IPAR_INFEAS_GENERIC_NAMES = 'MSK_OFF'`

MSK_IPAR_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"*

Groups *Overall solver*

Example `param.MSK_IPAR_INFEAS_PREFER_PRIMAL = 'MSK_ON'`

MSK_IPAR_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"*

Groups *Data input/output*, *Solution input/output*

Example `param.MSK_IPAR_INFEAS_REPORT_AUTO = 'MSK_OFF'`

MSK_IPAR_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*

Example `param.MSK_IPAR_INFEAS_REPORT_LEVEL = 1`

MSK_IPAR_INTPNT_BASIS

Controls whether the interior-point optimizer also computes an optimal basis.

Default *"ALWAYS"*

Accepted *"NEVER"*, *"ALWAYS"*, *"NO_ERROR"*, *"IF_FEASIBLE"*, *"RESERVERED"*

Groups *Interior-point method*, *Basis identification*

See also *MSK_IPAR_BI_IGNORE_MAX_ITER*, *MSK_IPAR_BI_IGNORE_NUM_ERROR*, *MSK_IPAR_BI_MAX_ITERATIONS*, *MSK_IPAR_BI_CLEAN_OPTIMIZER*

Example `param.MSK_IPAR_INTPNT_BASIS = 'MSK_BI_ALWAYS'`

MSK_IPAR_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*

Example `param.MSK_IPAR_INTPNT_DIFF_STEP = 'MSK_ON'`

MSK_IPAR_INTPNT_HOTSTART

Currently not in use.

Default *"NONE"*

Accepted *"NONE"*, *"PRIMAL"*, *"DUAL"*, *"PRIMAL_DUAL"*

Groups *Interior-point method*

Example `param.MSK_IPAR_INTPNT_HOTSTART = 'MSK_INTPNT_HOTSTART_NONE'`

`MSK_IPAR_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*

Example `param.MSK_IPAR_INTPNT_MAX_ITERATIONS = 400`

`MSK_IPAR_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*

Example `param.MSK_IPAR_INTPNT_MAX_NUM_COR = -1`

`MSK_IPAR_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*

Example `param.MSK_IPAR_INTPNT_MAX_NUM_REFINEMENT_STEPS = -1`

`MSK_IPAR_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"*

Groups *Overall system*

Example `param.MSK_IPAR_INTPNT_MULTI_THREAD = 'MSK_ON'`

`MSK_IPAR_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*

Example `param.MSK_IPAR_INTPNT_OFF_COL_TRH = 40`

`MSK_IPAR_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"*

Groups *Interior-point method*

Example `param.MSK_IPAR_INTPNT_ORDER_METHOD = 'MSK_ORDER_METHOD_FREE'`

`MSK_IPAR_INTPNT_PURIFY`
Currently not in use.

Default *"NONE"*
Accepted *"NONE"*, *"PRIMAL"*, *"DUAL"*, *"PRIMAL_DUAL"*, *"AUTO"*
Groups *Interior-point method*
Example `param.MSK_IPAR_INTPNT_PURIFY = 'MSK_PURIFY_NONE'`

`MSK_IPAR_INTPNT_REGULARIZATION_USE`
Controls whether regularization is allowed.

Default *"ON"*
Accepted *"ON"*, *"OFF"*
Groups *Interior-point method*
Example `param.MSK_IPAR_INTPNT_REGULARIZATION_USE = 'MSK_ON'`

`MSK_IPAR_INTPNT_SCALING`
Controls how the problem is scaled before the interior-point optimizer is used.

Default *"FREE"*
Accepted *"FREE"*, *"NONE"*, *"MODERATE"*, *"AGGRESSIVE"*
Groups *Interior-point method*
Example `param.MSK_IPAR_INTPNT_SCALING = 'MSK_SCALING_FREE'`

`MSK_IPAR_INTPNT_SOLVE_FORM`
Controls whether the primal or the dual problem is solved.

Default *"FREE"*
Accepted *"FREE"*, *"PRIMAL"*, *"DUAL"*
Groups *Interior-point method*
Example `param.MSK_IPAR_INTPNT_SOLVE_FORM = 'MSK_SOLVE_FREE'`

`MSK_IPAR_INTPNT_STARTING_POINT`
Starting point used by the interior-point optimizer.

Default *"FREE"*
Accepted *"FREE"*, *"GUESS"*, *"CONSTANT"*, *"SATISFY_BOUNDS"*
Groups *Interior-point method*
Example `param.MSK_IPAR_INTPNT_STARTING_POINT = 'MSK_STARTING_POINT_FREE'`

`MSK_IPAR_LICENSE_DEBUG`
This option is used to turn on debugging of the license manager.

Default *"OFF"*
Accepted *"ON"*, *"OFF"*
Groups *License manager*
Example `param.MSK_IPAR_LICENSE_DEBUG = 'MSK_OFF'`

`MSK_IPAR_LICENSE_PAUSE_TIME`
If *MSK_IPAR_LICENSE_WAIT* is *"MSK_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*
Example `param.MSK_IPAR_LICENSE_PAUSE_TIME = 100`

MSK_IPAR_LICENSE_SUPPRESS_EXPIRE_WRNS
: Controls whether license features expire warnings are suppressed.

  Default *"OFF"*

  Accepted *"ON"*, *"OFF"*

  Groups *License manager*, *Output information*

  Example `param.MSK_IPAR_LICENSE_SUPPRESS_EXPIRE_WRNS = 'MSK_OFF'`

MSK_IPAR_LICENSE_TRH_EXPIRY_WRN
: If a license feature expires in a numbers of days less than the value of this parameter then a warning will be issued.

  Default 7

  Accepted [0; +inf]

  Groups *License manager*, *Output information*

  Example `param.MSK_IPAR_LICENSE_TRH_EXPIRY_WRN = 7`

MSK_IPAR_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"*

  Groups *Overall solver*, *Overall system*, *License manager*

  Example `param.MSK_IPAR_LICENSE_WAIT = 'MSK_OFF'`

MSK_IPAR_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 *MSK_IPAR_LOG_CUT_SECOND_OPT* for the second and any subsequent optimizations.

  Default 10

  Accepted [0; +inf]

  Groups *Output information*, *Logging*

  See also *MSK_IPAR_LOG_CUT_SECOND_OPT*

  Example `param.MSK_IPAR_LOG = 10`

MSK_IPAR_LOG_ANA_PRO
: Controls amount of output from the problem analyzer.

  Default 1

  Accepted [0; +inf]

  Groups *Analysis*, *Logging*

  Example `param.MSK_IPAR_LOG_ANA_PRO = 1`

MSK_IPAR_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*

  Example `param.MSK_IPAR_LOG_BI = 1`

MSK_IPAR_LOG_BI_FREQ
: Controls how frequently the optimizer outputs information about the basis identification and how frequent the user-defined callback function is called.

Default 2500

Accepted [0; +inf]

Groups *Basis identification*, *Output information*, *Logging*

Example `param.MSK_IPAR_LOG_BI_FREQ = 2500`

`MSK_IPAR_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*

Example `param.MSK_IPAR_LOG_CHECK_CONVEXITY = 0`

`MSK_IPAR_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 *MSK_IPAR_LOG* and *MSK_IPAR_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 *MSK_IPAR_LOG*, *MSK_IPAR_LOG_INTPNT*, *MSK_IPAR_LOG_MIO*, *MSK_IPAR_LOG_SIM*

Example `param.MSK_IPAR_LOG_CUT_SECOND_OPT = 1`

`MSK_IPAR_LOG_EXPAND`

Controls the amount of logging when a data item such as the maximum number constrains is expanded.

Default 0

Accepted [0; +inf]

Groups *Output information*, *Logging*

Example `param.MSK_IPAR_LOG_EXPAND = 0`

`MSK_IPAR_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*

Example `param.MSK_IPAR_LOG_FEAS_REPAIR = 1`

`MSK_IPAR_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*

Example `param.MSK_IPAR_LOG_FILE = 1`

`MSK_IPAR_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*

Example `param.MSK_IPAR_LOG_INFEAS_ANA = 1`

`MSK_IPAR_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*

Example `param.MSK_IPAR_LOG_INTPNT = 1`

`MSK_IPAR_LOG_LOCAL_INFO`

Controls whether local identifying information like environment variables, filenames, IP addresses etc. are printed to the log.

Note that this will only affect some functions. Some functions that specifically emit system information will not be affected.

Default *"ON"*

Accepted *"ON"*, *"OFF"*

Groups *Output information*, *Logging*

Example `param.MSK_IPAR_LOG_LOCAL_INFO = 'MSK_ON'`

`MSK_IPAR_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*

Example `param.MSK_IPAR_LOG_MIO = 4`

`MSK_IPAR_LOG_MIO_FREQ`

Controls how frequent the mixed-integer optimizer prints the log line. It will print line every time *MSK_IPAR_LOG_MIO_FREQ* relaxations have been solved.

Default 10

Accepted [-inf; +inf]

Groups *Mixed-integer optimization*, *Output information*, *Logging*

Example `param.MSK_IPAR_LOG_MIO_FREQ = 10`

`MSK_IPAR_LOG_ORDER`

If turned on, then factor lines are added to the log.

Default 1

Accepted [0; +inf]

Groups *Output information*, *Logging*

Example `param.MSK_IPAR_LOG_ORDER = 1`

`MSK_IPAR_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*

Example `param.MSK_IPAR_LOG_PRESOLVE = 1`

MSK_IPAR_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*

Example `param.MSK_IPAR_LOG_RESPONSE = 0`

MSK_IPAR_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*

Example `param.MSK_IPAR_LOG_SENSITIVITY = 1`

MSK_IPAR_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*

Example `param.MSK_IPAR_LOG_SENSITIVITY_OPT = 0`

MSK_IPAR_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*

Example `param.MSK_IPAR_LOG_SIM = 4`

MSK_IPAR_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*

Example `param.MSK_IPAR_LOG_SIM_FREQ = 1000`

MSK_IPAR_LOG_SIM_MINOR

Currently not in use.

Default 1

Accepted [0; +inf]

Groups *Simplex optimizer*, *Output information*

Example `param.MSK_IPAR_LOG_SIM_MINOR = 1`

MSK_IPAR_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*

Example `param.MSK_IPAR_LOG_STORAGE = 0`

`MSK_IPAR_MAX_NUM_WARNINGS`

Each warning is shown a limited number of times controlled by this parameter. A negative value is identical to infinite number of times.

Default 10

Accepted [-inf; +inf]

Groups *Output information*

Example `param.MSK_IPAR_MAX_NUM_WARNINGS = 10`

`MSK_IPAR_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"*

Groups *Mixed-integer optimization*

Example `param.MSK_IPAR_MIO_BRANCH_DIR = 'MSK_BRANCH_DIR_FREE'`

`MSK_IPAR_MIO_CONIC_OUTER_APPROXIMATION`

If this option is turned on outer approximation is used when solving relaxations of conic problems; otherwise interior point is used.

Default *"OFF"*

Accepted *"ON"*, *"OFF"*

Groups *Mixed-integer optimization*

Example `param.MSK_IPAR_MIO_CONIC_OUTER_APPROXIMATION = 'MSK_OFF'`

`MSK_IPAR_MIO_CUT_CLIQUE`

Controls whether clique cuts should be generated.

Default *"ON"*

Accepted *"ON"*, *"OFF"*

Groups *Mixed-integer optimization*

Example `param.MSK_IPAR_MIO_CUT_CLIQUE = 'MSK_ON'`

`MSK_IPAR_MIO_CUT_CMIR`

Controls whether mixed integer rounding cuts should be generated.

Default *"ON"*

Accepted *"ON"*, *"OFF"*

Groups *Mixed-integer optimization*

Example `param.MSK_IPAR_MIO_CUT_CMIR = 'MSK_ON'`

`MSK_IPAR_MIO_CUT_GMI`

Controls whether GMI cuts should be generated.

Default *"ON"*

Accepted *"ON"*, *"OFF"*

Groups *Mixed-integer optimization*

Example `param.MSK_IPAR_MIO_CUT_GMI = 'MSK_ON'`

`MSK_IPAR_MIO_CUT_IMPLIED_BOUND`

Controls whether implied bound cuts should be generated.

Default *"OFF"*

Accepted *"ON"*, *"OFF"*

Groups *Mixed-integer optimization*

Example `param.MSK_IPAR_MIO_CUT_IMPLIED_BOUND = 'MSK_OFF'`

`MSK_IPAR_MIO_CUT_KNAPSACK_COVER`

Controls whether knapsack cover cuts should be generated.

Default *"OFF"*

Accepted *"ON"*, *"OFF"*

Groups *Mixed-integer optimization*

Example `param.MSK_IPAR_MIO_CUT_KNAPSACK_COVER = 'MSK_OFF'`

`MSK_IPAR_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*

Example `param.MSK_IPAR_MIO_CUT_SELECTION_LEVEL = -1`

`MSK_IPAR_MIO_FEASPUMP_LEVEL`

Controls the way the Feasibility Pump heuristic is employed by the mixed-integer optimizer.

- -1. The optimizer chooses how the Feasibility Pump is used
- 0. The Feasibility Pump is disabled
- 1. The Feasibility Pump is enabled with an effort to improve solution quality
- 2. The Feasibility Pump is enabled with an effort to reach feasibility early

Default -1

Accepted [-1; 2]

Groups *Mixed-integer optimization*

Example `param.MSK_IPAR_MIO_FEASPUMP_LEVEL = -1`

`MSK_IPAR_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*

Example `param.MSK_IPAR_MIO_HEURISTIC_LEVEL = -1`

`MSK_IPAR_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*

Example `param.MSK_IPAR_MIO_MAX_NUM_BRANCHES = -1`

MSK_IPAR_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*

Example `param.MSK_IPAR_MIO_MAX_NUM_RELAXS = -1`

MSK_IPAR_MIO_MAX_NUM_ROOT_CUT_ROUNDS

Maximum number of cut separation rounds at the root node.

Default 100

Accepted [0; +inf]

Groups *Mixed-integer optimization*, *Termination criteria*

Example `param.MSK_IPAR_MIO_MAX_NUM_ROOT_CUT_ROUNDS = 100`

MSK_IPAR_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*

Example `param.MSK_IPAR_MIO_MAX_NUM_SOLUTIONS = -1`

MSK_IPAR_MIO_MODE

Controls whether the optimizer includes the integer restrictions when solving a (mixed) integer optimization problem.

Default *"SATISFIED"*

Accepted *"IGNORED"*, *"SATISFIED"*

Groups *Overall solver*

Example `param.MSK_IPAR_MIO_MODE = 'MSK_MIO_MODE_SATISFIED'`

MSK_IPAR_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"*

Groups *Mixed-integer optimization*

Example `param.MSK_IPAR_MIO_NODE_OPTIMIZER = 'MSK_OPTIMIZER_FREE'`

MSK_IPAR_MIO_NODE_SELECTION

Controls the node selection strategy employed by the mixed-integer optimizer.

Default *"FREE"*

Accepted *"FREE"*, *"FIRST"*, *"BEST"*, *"PSEUDO"*

Groups *Mixed-integer optimization*

Example `param.MSK_IPAR_MIO_NODE_SELECTION = 'MSK_MIO_NODE_SELECTION_FREE'`

MSK_IPAR_MIO_PERSPECTIVE_REFORMULATE

Enables or disables perspective reformulation in presolve.

Default *"ON"*

Accepted *"ON"*, *"OFF"*

Groups *Mixed-integer optimization*

Example `param.MSK_IPAR_MIO_PERSPECTIVE_REFORMULATE = 'MSK_ON'`

MSK_IPAR_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*
Example `param.MSK_IPAR_MIO_PROBING_LEVEL = -1`

MSK_IPAR_MIO_PROPAGATE_OBJECTIVE_CONSTRAINT
Use objective domain propagation.

Default *"OFF"*
Accepted *"ON"*, *"OFF"*
Groups *Mixed-integer optimization*
Example `param.MSK_IPAR_MIO_PROPAGATE_OBJECTIVE_CONSTRAINT = 'MSK_OFF'`

MSK_IPAR_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*
Example `param.MSK_IPAR_MIO_RINS_MAX_NODES = -1`

MSK_IPAR_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"*
Groups *Mixed-integer optimization*
Example `param.MSK_IPAR_MIO_ROOT_OPTIMIZER = 'MSK_OPTIMIZER_FREE'`

MSK_IPAR_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*
Example `param.MSK_IPAR_MIO_ROOT_REPEAT_PRESOLVE_LEVEL = -1`

MSK_IPAR_MIO_SEED
Sets the random seed used for randomization in the mixed integer optimizer. Selecting a different seed can change the path the optimizer takes to the optimal solution.

Default 42
Accepted [0; +inf]

Groups *Mixed-integer optimization*

Example `param.MSK_IPAR_MIO_SEED = 42`

`MSK_IPAR_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*

Example `param.MSK_IPAR_MIO_VB_DETECTION_LEVEL = -1`

`MSK_IPAR_MT_SPINCOUNT`

Set the number of iterations to spin before sleeping.

Default 0

Accepted [0; 1000000000]

Groups *Overall system*

Example `param.MSK_IPAR_MT_SPINCOUNT = 0`

`MSK_IPAR_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.

The value of this parameter set at first optimization remains constant through the lifetime of the process. **MOSEK** will allocate a thread pool of given size, and changing the parameter value later will have no effect. It will, however, remain possible to demand single-threaded execution by setting *MSK_IPAR_INTPNT_MULTI_THREAD*.

Default 0

Accepted [0; +inf]

Groups *Overall system*

Example `param.MSK_IPAR_NUM_THREADS = 0`

`MSK_IPAR_OPF_WRITE_HEADER`

Write a text header with date and **MOSEK** version in an OPF file.

Default *"ON"*

Accepted *"ON"*, *"OFF"*

Groups *Data input/output*

Example `param.MSK_IPAR_OPF_WRITE_HEADER = 'MSK_ON'`

`MSK_IPAR_OPF_WRITE_HINTS`

Write a hint section with problem dimensions in the beginning of an OPF file.

Default *"ON"*

Accepted *"ON"*, *"OFF"*

Groups *Data input/output*

Example `param.MSK_IPAR_OPF_WRITE_HINTS = 'MSK_ON'`

`MSK_IPAR_OPF_WRITE_LINE_LENGTH`

Aim to keep lines in OPF files not much longer than this.

Default 80

Accepted [0; +inf]

Groups *Data input/output*

Example `param.MSK_IPAR_OPF_WRITE_LINE_LENGTH = 80`

`MSK_IPAR_OPF_WRITE_PARAMETERS`

Write a parameter section in an OPF file.

Default *"OFF"*

Accepted *"ON"*, *"OFF"*

Groups *Data input/output*

Example `param.MSK_IPAR_OPF_WRITE_PARAMETERS = 'MSK_OFF'`

`MSK_IPAR_OPF_WRITE_PROBLEM`

Write objective, constraints, bounds etc. to an OPF file.

Default *"ON"*

Accepted *"ON"*, *"OFF"*

Groups *Data input/output*

Example `param.MSK_IPAR_OPF_WRITE_PROBLEM = 'MSK_ON'`

`MSK_IPAR_OPF_WRITE_SOL_BAS`

If *MSK_IPAR_OPF_WRITE_SOLUTIONS* is *"MSK_ON"* and a basic solution is defined, include the basic solution in OPF files.

Default *"ON"*

Accepted *"ON"*, *"OFF"*

Groups *Data input/output*

Example `param.MSK_IPAR_OPF_WRITE_SOL_BAS = 'MSK_ON'`

`MSK_IPAR_OPF_WRITE_SOL_ITG`

If *MSK_IPAR_OPF_WRITE_SOLUTIONS* is *"MSK_ON"* and an integer solution is defined, write the integer solution in OPF files.

Default *"ON"*

Accepted *"ON"*, *"OFF"*

Groups *Data input/output*

Example `param.MSK_IPAR_OPF_WRITE_SOL_ITG = 'MSK_ON'`

`MSK_IPAR_OPF_WRITE_SOL_ITR`

If *MSK_IPAR_OPF_WRITE_SOLUTIONS* is *"MSK_ON"* and an interior solution is defined, write the interior solution in OPF files.

Default *"ON"*

Accepted *"ON"*, *"OFF"*

Groups *Data input/output*

Example `param.MSK_IPAR_OPF_WRITE_SOL_ITR = 'MSK_ON'`

`MSK_IPAR_OPF_WRITE_SOLUTIONS`

Enable inclusion of solutions in the OPF files.

Default *"OFF"*

Accepted *"ON"*, *"OFF"*

Groups *Data input/output*

Example `param.MSK_IPAR_OPF_WRITE_SOLUTIONS = 'MSK_OFF'`

`MSK_IPAR_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"*

Groups *Overall solver*

Example `param.MSK_IPAR_OPTIMIZER = 'MSK_OPTIMIZER_FREE'`

`MSK_IPAR_PARAM_READ_CASE_NAME`

If turned on, then names in the parameter file are case sensitive.

Default *"ON"*

Accepted *"ON"*, *"OFF"*

Groups *Data input/output*

Example `param.MSK_IPAR_PARAM_READ_CASE_NAME = 'MSK_ON'`

`MSK_IPAR_PARAM_READ_IGN_ERROR`

If turned on, then errors in parameter settings is ignored.

Default *"OFF"*

Accepted *"ON"*, *"OFF"*

Groups *Data input/output*

Example `param.MSK_IPAR_PARAM_READ_IGN_ERROR = 'MSK_OFF'`

`MSK_IPAR_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*

Example `param.MSK_IPAR_PRESOLVE_ELIMINATOR_MAX_FILL = -1`

`MSK_IPAR_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*

Example `param.MSK_IPAR_PRESOLVE_ELIMINATOR_MAX_NUM_TRIES = -1`

`MSK_IPAR_PRESOLVE_LEVEL`

Currently not used.

Default -1

Accepted [-inf; +inf]

Groups *Overall solver*, *Presolve*

Example `param.MSK_IPAR_PRESOLVE_LEVEL = -1`

`MSK_IPAR_PRESOLVE_LINDEP_ABS_WORK_TRH`

Controls linear dependency check in presolve. The linear dependency check is potentially computationally expensive.

Default 100

Accepted [-inf; +inf]

Groups *Presolve*

Example `param.MSK_IPAR_PRESOLVE_LINDEP_ABS_WORK_TRH = 100`

`MSK_IPAR_PRESOLVE_LINDEP_REL_WORK_TRH`

Controls linear dependency check in presolve. The linear dependency check is potentially computationally expensive.

Default 100

Accepted [-inf; +inf]

Groups *Presolve*

Example param.MSK_IPAR_PRESOLVE_LINDEP_REL_WORK_TRH = 100

MSK_IPAR_PRESOLVE_LINDEP_USE
Controls whether the linear constraints are checked for linear dependencies.

Default *"ON"*
Accepted *"ON"*, *"OFF"*
Groups *Presolve*
Example param.MSK_IPAR_PRESOLVE_LINDEP_USE = 'MSK_ON'

MSK_IPAR_PRESOLVE_MAX_NUM_PASS
Control the maximum number of times presolve passes over the problem. A negative value implies **MOSEK** decides.

Default -1
Accepted [-inf; +inf]
Groups *Presolve*
Example param.MSK_IPAR_PRESOLVE_MAX_NUM_PASS = -1

MSK_IPAR_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*
Example param.MSK_IPAR_PRESOLVE_MAX_NUM_REDUCTIONS = -1

MSK_IPAR_PRESOLVE_USE
Controls whether the presolve is applied to a problem before it is optimized.

Default *"FREE"*
Accepted *"OFF"*, *"ON"*, *"FREE"*
Groups *Overall solver*, *Presolve*
Example param.MSK_IPAR_PRESOLVE_USE = 'MSK_PRESOLVE_MODE_FREE'

MSK_IPAR_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"*
Groups *Overall solver*
Example param.MSK_IPAR_PRIMAL_REPAIR_OPTIMIZER = 'MSK_OPTIMIZER_FREE'

MSK_IPAR_PTF_WRITE_TRANSFORM
If *MSK_IPAR_PTF_WRITE_TRANSFORM* is *"MSK_ON"*, constraint blocks with identifiable conic slacks are transformed into conic constraints and the slacks are eliminated.

Default *"ON"*
Accepted *"ON"*, *"OFF"*
Groups *Data input/output*
Example param.MSK_IPAR_PTF_WRITE_TRANSFORM = 'MSK_ON'

MSK_IPAR_READ_DEBUG
Turns on additional debugging information when reading files.

Default *"OFF"*
Accepted *"ON"*, *"OFF"*

Groups *Data input/output*

Example `param.MSK_IPAR_READ_DEBUG = 'MSK_OFF'`

`MSK_IPAR_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*

Example `param.MSK_IPAR_READ_KEEP_FREE_CON = 'MSK_OFF'`

`MSK_IPAR_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"*

Groups *Data input/output*

Example `param.MSK_IPAR_READ_LP_DROP_NEW_VARS_IN_BOU = 'MSK_OFF'`

`MSK_IPAR_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"*

Groups *Data input/output*

Example `param.MSK_IPAR_READ_LP_QUOTED_NAMES = 'MSK_ON'`

`MSK_IPAR_READ_MPS_FORMAT`

Controls how strictly the MPS file reader interprets the MPS format.

Default *"FREE"*

Accepted *"STRICT"*, *"RELAXED"*, *"FREE"*, *"CPLEX"*

Groups *Data input/output*

Example `param.MSK_IPAR_READ_MPS_FORMAT = 'MSK_MPS_FORMAT_FREE'`

`MSK_IPAR_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*

Example `param.MSK_IPAR_READ_MPS_WIDTH = 1024`

`MSK_IPAR_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"*

Groups *Data input/output*

Example `param.MSK_IPAR_READ_TASK_IGNORE_PARAM = 'MSK_OFF'`

`MSK_IPAR_REMOVE_UNUSED_SOLUTIONS`

Removes unused solutions before the optimization is performed.

Default *"OFF"*

Accepted *"ON"*, *"OFF"*

Groups *Overall system*

Example `param.MSK_IPAR_REMOVE_UNUSED_SOLUTIONS = 'MSK_OFF'`

`MSK_IPAR_SENSITIVITY_ALL`

Not applicable.

Default *"OFF"*

Accepted *"ON"*, *"OFF"*

Groups *Overall solver*

Example `param.MSK_IPAR_SENSITIVITY_ALL = 'MSK_OFF'`

`MSK_IPAR_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"*

Groups *Overall solver*, *Simplex optimizer*

Example `param.MSK_IPAR_SENSITIVITY_OPTIMIZER = 'MSK_OPTIMIZER_FREE_SIMPLEX'`

`MSK_IPAR_SENSITIVITY_TYPE`

Controls which type of sensitivity analysis is to be performed.

Default *"BASIS"*

Accepted *"BASIS"*

Groups *Overall solver*

Example `param.MSK_IPAR_SENSITIVITY_TYPE = 'MSK_SENSITIVITY_TYPE_BASIS'`

`MSK_IPAR_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"*

Groups *Simplex optimizer*

Example `param.MSK_IPAR_SIM_BASIS_FACTOR_USE = 'MSK_ON'`

`MSK_IPAR_SIM_DEGEN`

Controls how aggressively degeneration is handled.

Default *"FREE"*

Accepted *"NONE"*, *"FREE"*, *"AGGRESSIVE"*, *"MODERATE"*, *"MINIMUM"*

Groups *Simplex optimizer*

Example `param.MSK_IPAR_SIM_DEGEN = 'MSK_SIM_DEGEN_FREE'`

`MSK_IPAR_SIM_DUAL_CRASH`

Controls whether crashing is performed in the dual simplex optimizer. If this parameter is set to x, then a crash will be performed if a basis consists of more than $(100 - x) \mod f_v$ entries, where f_v is the number of fixed variables.

Default 90

Accepted [0; +inf]

Groups *Dual simplex*

Example `param.MSK_IPAR_SIM_DUAL_CRASH = 90`

`MSK_IPAR_SIM_DUAL_PHASEONE_METHOD`

An experimental feature.

Default 0

Accepted [0; 10]

Groups *Simplex optimizer*

Example `param.MSK_IPAR_SIM_DUAL_PHASEONE_METHOD = 0`

`MSK_IPAR_SIM_DUAL_RESTRICT_SELECTION`

The dual simplex optimizer can use a so-called restricted selection/pricing strategy to choose the outgoing variable. Hence, if restricted selection is applied, then the dual simplex optimizer first choose a subset of all the potential outgoing variables. Next, for some time it will choose the outgoing variable only among the subset. From time to time the subset is redefined. A larger value of this parameter implies that the optimizer will be more aggressive in its restriction strategy, i.e. a value of 0 implies that the restriction strategy is not applied at all.

Default 50

Accepted [0; 100]

Groups *Dual simplex*

Example `param.MSK_IPAR_SIM_DUAL_RESTRICT_SELECTION = 50`

`MSK_IPAR_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"*

Groups *Dual simplex*

Example `param.MSK_IPAR_SIM_DUAL_SELECTION = 'MSK_SIM_SELECTION_FREE'`

`MSK_IPAR_SIM_EXPLOIT_DUPVEC`

Controls if the simplex optimizers are allowed to exploit duplicated columns.

Default *"OFF"*

Accepted *"ON"*, *"OFF"*, *"FREE"*

Groups *Simplex optimizer*

Example `param.MSK_IPAR_SIM_EXPLOIT_DUPVEC = 'MSK_SIM_EXPLOIT_DUPVEC_OFF'`

`MSK_IPAR_SIM_HOTSTART`

Controls the type of hot-start that the simplex optimizer perform.

Default *"FREE"*

Accepted *"NONE"*, *"FREE"*, *"STATUS_KEYS"*

Groups *Simplex optimizer*

Example `param.MSK_IPAR_SIM_HOTSTART = 'MSK_SIM_HOTSTART_FREE'`

`MSK_IPAR_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*

Example `param.MSK_IPAR_SIM_HOTSTART_LU = 'MSK_ON'`

`MSK_IPAR_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*

Example `param.MSK_IPAR_SIM_MAX_ITERATIONS = 10000000`

MSK_IPAR_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*

Example `param.MSK_IPAR_SIM_MAX_NUM_SETBACKS = 250`

MSK_IPAR_SIM_NON_SINGULAR

Controls if the simplex optimizer ensures a non-singular basis, if possible.

Default *"ON"*

Accepted *"ON"*, *"OFF"*

Groups *Simplex optimizer*

Example `param.MSK_IPAR_SIM_NON_SINGULAR = 'MSK_ON'`

MSK_IPAR_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*

Example `param.MSK_IPAR_SIM_PRIMAL_CRASH = 90`

MSK_IPAR_SIM_PRIMAL_PHASEONE_METHOD

An experimental feature.

Default 0

Accepted [0; 10]

Groups *Simplex optimizer*

Example `param.MSK_IPAR_SIM_PRIMAL_PHASEONE_METHOD = 0`

MSK_IPAR_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*

Example `param.MSK_IPAR_SIM_PRIMAL_RESTRICT_SELECTION = 50`

MSK_IPAR_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"*

Groups *Primal simplex*

Example `param.MSK_IPAR_SIM_PRIMAL_SELECTION = 'MSK_SIM_SELECTION_FREE'`

MSK_IPAR_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*
Example `param.MSK_IPAR_SIM_REFACTOR_FREQ = 0`

`MSK_IPAR_SIM_REFORMULATION`
Controls if the simplex optimizers are allowed to reformulate the problem.

Default *"OFF"*
Accepted *"ON"*, *"OFF"*, *"FREE"*, *"AGGRESSIVE"*
Groups *Simplex optimizer*
Example `param.MSK_IPAR_SIM_REFORMULATION = 'MSK_SIM_REFORMULATION_OFF'`

`MSK_IPAR_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"*
Groups *Simplex optimizer*
Example `param.MSK_IPAR_SIM_SAVE_LU = 'MSK_OFF'`

`MSK_IPAR_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"*
Groups *Simplex optimizer*
Example `param.MSK_IPAR_SIM_SCALING = 'MSK_SCALING_FREE'`

`MSK_IPAR_SIM_SCALING_METHOD`
Controls how the problem is scaled before a simplex optimizer is used.

Default *"POW2"*
Accepted *"POW2"*, *"FREE"*
Groups *Simplex optimizer*
Example `param.MSK_IPAR_SIM_SCALING_METHOD = 'MSK_SCALING_METHOD_POW2'`

`MSK_IPAR_SIM_SEED`
Sets the random seed used for randomization in the simplex optimizers.

Default 23456
Accepted [0; 32749]
Groups *Simplex optimizer*
Example `param.MSK_IPAR_SIM_SEED = 23456`

`MSK_IPAR_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"*
Groups *Simplex optimizer*
Example `param.MSK_IPAR_SIM_SOLVE_FORM = 'MSK_SOLVE_FREE'`

`MSK_IPAR_SIM_STABILITY_PRIORITY`
Controls how high priority the numerical stability should be given.

Default 50
Accepted [0; 100]
Groups *Simplex optimizer*

Example `param.MSK_IPAR_SIM_STABILITY_PRIORITY = 50`

`MSK_IPAR_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"*

Groups *Simplex optimizer*

Example `param.MSK_IPAR_SIM_SWITCH_OPTIMIZER = 'MSK_OFF'`

`MSK_IPAR_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"*

Groups *Solution input/output*

Example `param.MSK_IPAR_SOL_FILTER_KEEP_BASIC = 'MSK_OFF'`

`MSK_IPAR_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"*

Groups *Solution input/output*

Example `param.MSK_IPAR_SOL_FILTER_KEEP_RANGED = 'MSK_OFF'`

`MSK_IPAR_SOL_READ_NAME_WIDTH`

When a solution is read by **MOSEK** and some constraint, variable or cone names contain blanks, then a maximum name width much be specified. A negative value implies that no name contain blanks.

Default -1

Accepted [-inf; +inf]

Groups *Data input/output*, *Solution input/output*

Example `param.MSK_IPAR_SOL_READ_NAME_WIDTH = -1`

`MSK_IPAR_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*

Example `param.MSK_IPAR_SOL_READ_WIDTH = 1024`

`MSK_IPAR_SOLUTION_CALLBACK`

Indicates whether solution callbacks will be performed during the optimization.

Default *"OFF"*

Accepted *"ON"*, *"OFF"*

Groups *Progress callback*, *Overall solver*

Example `param.MSK_IPAR_SOLUTION_CALLBACK = 'MSK_OFF'`

`MSK_IPAR_TIMING_LEVEL`

Controls the amount of timing performed inside **MOSEK**.

Default 1

Accepted [0; +inf]

Groups *Overall system*

Example `param.MSK_IPAR_TIMING_LEVEL = 1`

`MSK_IPAR_WRITE_BAS_CONSTRAINTS`

Controls whether the constraint section is written to the basic solution file.

Default *"ON"*

Accepted *"ON"*, *"OFF"*

Groups *Data input/output*, *Solution input/output*

Example `param.MSK_IPAR_WRITE_BAS_CONSTRAINTS = 'MSK_ON'`

`MSK_IPAR_WRITE_BAS_HEAD`

Controls whether the header section is written to the basic solution file.

Default *"ON"*

Accepted *"ON"*, *"OFF"*

Groups *Data input/output*, *Solution input/output*

Example `param.MSK_IPAR_WRITE_BAS_HEAD = 'MSK_ON'`

`MSK_IPAR_WRITE_BAS_VARIABLES`

Controls whether the variables section is written to the basic solution file.

Default *"ON"*

Accepted *"ON"*, *"OFF"*

Groups *Data input/output*, *Solution input/output*

Example `param.MSK_IPAR_WRITE_BAS_VARIABLES = 'MSK_ON'`

`MSK_IPAR_WRITE_COMPRESSION`

Controls whether the data file is compressed while it is written. 0 means no compression while higher values mean more compression.

Default 9

Accepted [0; +inf]

Groups *Data input/output*

Example `param.MSK_IPAR_WRITE_COMPRESSION = 9`

`MSK_IPAR_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"*

Groups *Data input/output*

Example `param.MSK_IPAR_WRITE_DATA_PARAM = 'MSK_OFF'`

`MSK_IPAR_WRITE_FREE_CON`

Controls whether the free constraints are written to the data file.

Default *"ON"*

Accepted *"ON"*, *"OFF"*

Groups *Data input/output*

Example `param.MSK_IPAR_WRITE_FREE_CON = 'MSK_ON'`

`MSK_IPAR_WRITE_GENERIC_NAMES`

Controls whether generic names should be used instead of user-defined names when writing to the data file.

Default *"OFF"*

Accepted *"ON"*, *"OFF"*

Groups *Data input/output*

Example `param.MSK_IPAR_WRITE_GENERIC_NAMES = 'MSK_OFF'`

`MSK_IPAR_WRITE_GENERIC_NAMES_IO`

Index origin used in generic names.

Default 1

Accepted [0; +inf]

Groups *Data input/output*

Example `param.MSK_IPAR_WRITE_GENERIC_NAMES_IO = 1`

`MSK_IPAR_WRITE_IGNORE_INCOMPATIBLE_ITEMS`

Controls if the writer ignores incompatible problem items when writing files.

Default *"OFF"*

Accepted

- *"ON"*: Ignore items that cannot be written to the current output file format.
- *"OFF"*: Produce an error if the problem contains items that cannot the written to the current output file format.

Groups *Data input/output*

Example `param.MSK_IPAR_WRITE_IGNORE_INCOMPATIBLE_ITEMS = 'MSK_OFF'`

`MSK_IPAR_WRITE_INT_CONSTRAINTS`

Controls whether the constraint section is written to the integer solution file.

Default *"ON"*

Accepted *"ON"*, *"OFF"*

Groups *Data input/output*, *Solution input/output*

Example `param.MSK_IPAR_WRITE_INT_CONSTRAINTS = 'MSK_ON'`

`MSK_IPAR_WRITE_INT_HEAD`

Controls whether the header section is written to the integer solution file.

Default *"ON"*

Accepted *"ON"*, *"OFF"*

Groups *Data input/output*, *Solution input/output*

Example `param.MSK_IPAR_WRITE_INT_HEAD = 'MSK_ON'`

`MSK_IPAR_WRITE_INT_VARIABLES`

Controls whether the variables section is written to the integer solution file.

Default *"ON"*

Accepted *"ON"*, *"OFF"*

Groups *Data input/output*, *Solution input/output*

Example `param.MSK_IPAR_WRITE_INT_VARIABLES = 'MSK_ON'`

`MSK_IPAR_WRITE_LP_FULL_OBJ`

Write all variables, including the ones with 0-coefficients, in the objective.

Default *"ON"*

Accepted *"ON"*, *"OFF"*

Groups *Data input/output*

Example `param.MSK_IPAR_WRITE_LP_FULL_OBJ = 'MSK_ON'`

`MSK_IPAR_WRITE_LP_LINE_WIDTH`

Maximum width of line in an LP file written by **MOSEK**.

Default 80

Accepted [40; +inf]

Groups *Data input/output*

Example `param.MSK_IPAR_WRITE_LP_LINE_WIDTH = 80`

`MSK_IPAR_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"*

Groups *Data input/output*

Example `param.MSK_IPAR_WRITE_LP_QUOTED_NAMES = 'MSK_ON'`

`MSK_IPAR_WRITE_LP_STRICT_FORMAT`

Controls whether LP output files satisfy the LP format strictly.

Default *"OFF"*

Accepted *"ON"*, *"OFF"*

Groups *Data input/output*

Example `param.MSK_IPAR_WRITE_LP_STRICT_FORMAT = 'MSK_OFF'`

`MSK_IPAR_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*

Example `param.MSK_IPAR_WRITE_LP_TERMS_PER_LINE = 10`

`MSK_IPAR_WRITE_MPS_FORMAT`

Controls in which format the MPS is written.

Default *"FREE"*

Accepted *"STRICT"*, *"RELAXED"*, *"FREE"*, *"CPLEX"*

Groups *Data input/output*

Example `param.MSK_IPAR_WRITE_MPS_FORMAT = 'MSK_MPS_FORMAT_FREE'`

`MSK_IPAR_WRITE_MPS_INT`

Controls if marker records are written to the MPS file to indicate whether variables are integer restricted.

Default *"ON"*

Accepted *"ON"*, *"OFF"*

Groups *Data input/output*

Example `param.MSK_IPAR_WRITE_MPS_INT = 'MSK_ON'`

`MSK_IPAR_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*

Example `param.MSK_IPAR_WRITE_PRECISION = 15`

`MSK_IPAR_WRITE_SOL_BARVARIABLES`

Controls whether the symmetric matrix variables section is written to the solution file.

Default *"ON"*

Accepted *"ON"*, *"OFF"*

Groups *Data input/output*, *Solution input/output*

Example `param.MSK_IPAR_WRITE_SOL_BARVARIABLES = 'MSK_ON'`

MSK_IPAR_WRITE_SOL_CONSTRAINTS
Controls whether the constraint section is written to the solution file.

Default *"ON"*
Accepted *"ON"*, *"OFF"*
Groups *Data input/output*, *Solution input/output*
Example `param.MSK_IPAR_WRITE_SOL_CONSTRAINTS = 'MSK_ON'`

MSK_IPAR_WRITE_SOL_HEAD
Controls whether the header section is written to the solution file.

Default *"ON"*
Accepted *"ON"*, *"OFF"*
Groups *Data input/output*, *Solution input/output*
Example `param.MSK_IPAR_WRITE_SOL_HEAD = 'MSK_ON'`

MSK_IPAR_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"*
Groups *Data input/output*, *Solution input/output*
Example `param.MSK_IPAR_WRITE_SOL_IGNORE_INVALID_NAMES = 'MSK_OFF'`

MSK_IPAR_WRITE_SOL_VARIABLES
Controls whether the variables section is written to the solution file.

Default *"ON"*
Accepted *"ON"*, *"OFF"*
Groups *Data input/output*, *Solution input/output*
Example `param.MSK_IPAR_WRITE_SOL_VARIABLES = 'MSK_ON'`

MSK_IPAR_WRITE_TASK_INC_SOL
Controls whether the solutions are stored in the task file too.

Default *"ON"*
Accepted *"ON"*, *"OFF"*
Groups *Data input/output*
Example `param.MSK_IPAR_WRITE_TASK_INC_SOL = 'MSK_ON'`

MSK_IPAR_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"*
Groups *Data input/output*
Example `param.MSK_IPAR_WRITE_XML_MODE = 'MSK_WRITE_XML_MODE_ROW'`

15.5.3 String parameters

sparam
The enumeration type containing all string parameters.

MSK_SPAR_BAS_SOL_FILE_NAME
Name of the `bas` solution file.

Accepted Any valid file name.
Groups *Data input/output*, *Solution input/output*
Example `param.MSK_SPAR_BAS_SOL_FILE_NAME = 'somevalue'`

MSK_SPAR_DATA_FILE_NAME

Data are read and written to this file.

Accepted Any valid file name.

Groups *Data input/output*

Example `param.MSK_SPAR_DATA_FILE_NAME = 'somevalue'`

MSK_SPAR_DEBUG_FILE_NAME

MOSEK debug file.

Accepted Any valid file name.

Groups *Data input/output*

Example `param.MSK_SPAR_DEBUG_FILE_NAME = 'somevalue'`

MSK_SPAR_INT_SOL_FILE_NAME

Name of the `int` solution file.

Accepted Any valid file name.

Groups *Data input/output*, *Solution input/output*

Example `param.MSK_SPAR_INT_SOL_FILE_NAME = 'somevalue'`

MSK_SPAR_ITR_SOL_FILE_NAME

Name of the `itr` solution file.

Accepted Any valid file name.

Groups *Data input/output*, *Solution input/output*

Example `param.MSK_SPAR_ITR_SOL_FILE_NAME = 'somevalue'`

MSK_SPAR_MIO_DEBUG_STRING

For internal debugging purposes.

Accepted Any valid string.

Groups *Data input/output*

Example `param.MSK_SPAR_MIO_DEBUG_STRING = 'somevalue'`

MSK_SPAR_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*

Example `param.MSK_SPAR_PARAM_COMMENT_SIGN = '%%'`

MSK_SPAR_PARAM_READ_FILE_NAME

Modifications to the parameter database is read from this file.

Accepted Any valid file name.

Groups *Data input/output*

Example `param.MSK_SPAR_PARAM_READ_FILE_NAME = 'somevalue'`

MSK_SPAR_PARAM_WRITE_FILE_NAME

The parameter database is written to this file.

Accepted Any valid file name.

Groups *Data input/output*

Example `param.MSK_SPAR_PARAM_WRITE_FILE_NAME = 'somevalue'`

MSK_SPAR_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*

Example `param.MSK_SPAR_READ_MPS_BOU_NAME = 'somevalue'`

`MSK_SPAR_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*

Example `param.MSK_SPAR_READ_MPS_OBJ_NAME = 'somevalue'`

`MSK_SPAR_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*

Example `param.MSK_SPAR_READ_MPS_RAN_NAME = 'somevalue'`

`MSK_SPAR_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*

Example `param.MSK_SPAR_READ_MPS_RHS_NAME = 'somevalue'`

`MSK_SPAR_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*

Example `param.MSK_SPAR_REMOTE_ACCESS_TOKEN = 'somevalue'`

`MSK_SPAR_SENSITIVITY_FILE_NAME`

If defined, **MOSEK** 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*

Example `param.MSK_SPAR_SENSITIVITY_FILE_NAME = 'somevalue'`

`MSK_SPAR_SENSITIVITY_RES_FILE_NAME`

Accepted Any valid string.

Groups *Data input/output*

Example `param.MSK_SPAR_SENSITIVITY_RES_FILE_NAME = 'somevalue'`

`MSK_SPAR_SOL_FILTER_XC_LOW`

A filter used to determine which constraints should be listed in the solution file. A value of 0.5 means that all constraints having `xc[i]>0.5` should be listed, whereas `+0.5` means that all constraints having `xc[i]>=blc[i]+0.5` should be listed. An empty filter means that no filter is applied.

Accepted Any valid filter.

Groups *Data input/output*, *Solution input/output*

Example `param.MSK_SPAR_SOL_FILTER_XC_LOW = 'somevalue'`

`MSK_SPAR_SOL_FILTER_XC_UPR`

A filter used to determine which constraints should be listed in the solution file. A value of `0.5` means that all constraints having `xc[i]<0.5` should be listed, whereas `-0.5` means all constraints having `xc[i]<=buc[i]-0.5` should be listed. An empty filter means that no filter is applied.

Accepted Any valid filter.

Groups *Data input/output*, *Solution input/output*

Example `param.MSK_SPAR_SOL_FILTER_XC_UPR = 'somevalue'`

`MSK_SPAR_SOL_FILTER_XX_LOW`

A filter used to determine which variables should be listed in the solution file. A value of "0.5" means that all constraints having `xx[j]>=0.5` should be listed, whereas "+0.5" means that all constraints having `xx[j]>=blx[j]+0.5` should be listed. An empty filter means no filter is applied.

Accepted Any valid filter.

Groups *Data input/output*, *Solution input/output*

Example `param.MSK_SPAR_SOL_FILTER_XX_LOW = 'somevalue'`

`MSK_SPAR_SOL_FILTER_XX_UPR`

A filter used to determine which variables should be listed in the solution file. A value of "0.5" means that all constraints having `xx[j]<0.5` should be printed, whereas "-0.5" means all constraints having `xx[j]<=bux[j]-0.5` should be listed. An empty filter means no filter is applied.

Accepted Any valid file name.

Groups *Data input/output*, *Solution input/output*

Example `param.MSK_SPAR_SOL_FILTER_XX_UPR = 'somevalue'`

`MSK_SPAR_STAT_FILE_NAME`

Statistics file name.

Accepted Any valid file name.

Groups *Data input/output*

Example `param.MSK_SPAR_STAT_FILE_NAME = 'somevalue'`

`MSK_SPAR_STAT_KEY`

Key used when writing the summary file.

Accepted Any valid string.

Groups *Data input/output*

Example `param.MSK_SPAR_STAT_KEY = 'somevalue'`

`MSK_SPAR_STAT_NAME`

Name used when writing the statistics file.

Accepted Any valid XML string.

Groups *Data input/output*

Example `param.MSK_SPAR_STAT_NAME = 'somevalue'`

`MSK_SPAR_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*

Example `param.MSK_SPAR_WRITE_LP_GEN_VAR_NAME = 'xmskgen'`

15.6 Response codes

Response codes include:

- *Termination codes*
- *Warnings*
- *Errors*

The numerical code (in brackets) identifies the response in error messages and in the log output.

`rescode`
: The enumeration type containing all response codes.

15.6.1 Termination

`"MSK_RES_OK" (0)`
No error occurred.

`"MSK_RES_TRM_MAX_ITERATIONS" (10000)`
The optimizer terminated at the maximum number of iterations.

`"MSK_RES_TRM_MAX_TIME" (10001)`
The optimizer terminated at the maximum amount of time.

`"MSK_RES_TRM_OBJECTIVE_RANGE" (10002)`
The optimizer terminated with an objective value outside the objective range.

`"MSK_RES_TRM_MIO_NUM_RELAXS" (10008)`
The mixed-integer optimizer terminated as the maximum number of relaxations was reached.

`"MSK_RES_TRM_MIO_NUM_BRANCHES" (10009)`
The mixed-integer optimizer terminated as the maximum number of branches was reached.

`"MSK_RES_TRM_NUM_MAX_NUM_INT_SOLUTIONS" (10015)`
The mixed-integer optimizer terminated as the maximum number of feasible solutions was reached.

`"MSK_RES_TRM_STALL" (10006)`
The optimizer is terminated due to slow progress.

Stalling means that numerical problems prevent the optimizer from making reasonable progress and that it makes no sense to continue. In many cases this happens if the problem is badly scaled or otherwise ill-conditioned. There is no guarantee that the solution will be feasible or optimal. However, often stalling happens near the optimum, and the returned solution may be of good quality. Therefore, it is recommended to check the status of the solution. If the solution status is optimal the solution is most likely good enough for most practical purposes.

Please note that if a linear optimization problem is solved using the interior-point optimizer with basis identification turned on, the returned basic solution likely to have high accuracy, even though the optimizer stalled.

Some common causes of stalling are a) badly scaled models, b) near feasible or near infeasible problems.

`"MSK_RES_TRM_USER_CALLBACK" (10007)`
The optimizer terminated due to the return of the user-defined callback function.

`"MSK_RES_TRM_MAX_NUM_SETBACKS" (10020)`
The optimizer terminated as the maximum number of set-backs was reached. This indicates serious numerical problems and a possibly badly formulated problem.

`"MSK_RES_TRM_NUMERICAL_PROBLEM" (10025)`
The optimizer terminated due to numerical problems.

`"MSK_RES_TRM_INTERNAL" (10030)`
The optimizer terminated due to some internal reason. Please contact **MOSEK** support.

`"MSK_RES_TRM_INTERNAL_STOP" (10031)`
The optimizer terminated for internal reasons. Please contact **MOSEK** support.

15.6.2 Warnings

`"MSK_RES_WRN_OPEN_PARAM_FILE" (50)`
The parameter file could not be opened.

`"MSK_RES_WRN_LARGE_BOUND" (51)`
A numerically large bound value is specified.

`"MSK_RES_WRN_LARGE_LO_BOUND" (52)`
A numerically large lower bound value is specified.

`"MSK_RES_WRN_LARGE_UP_BOUND" (53)`
A numerically large upper bound value is specified.

`"MSK_RES_WRN_LARGE_CON_FX" (54)`
An equality constraint is fixed to a numerically large value. This can cause numerical problems.

"MSK_RES_WRN_LARGE_CJ" (57)
    A numerically large value is specified for one c_j.

"MSK_RES_WRN_LARGE_AIJ" (62)
    A numerically large value is specified for an $a_{i,j}$ element in A. The parameter *MSK_DPAR_DATA_TOL_AIJ_LARGE* controls when an $a_{i,j}$ is considered large.

"MSK_RES_WRN_ZERO_AIJ" (63)
    One or more zero elements are specified in A.

"MSK_RES_WRN_NAME_MAX_LEN" (65)
    A name is longer than the buffer that is supposed to hold it.

"MSK_RES_WRN_SPAR_MAX_LEN" (66)
    A value for a string parameter is longer than the buffer that is supposed to hold it.

"MSK_RES_WRN_MPS_SPLIT_RHS_VECTOR" (70)
    An RHS vector is split into several nonadjacent parts in an MPS file.

"MSK_RES_WRN_MPS_SPLIT_RAN_VECTOR" (71)
    A RANGE vector is split into several nonadjacent parts in an MPS file.

"MSK_RES_WRN_MPS_SPLIT_BOU_VECTOR" (72)
    A BOUNDS vector is split into several nonadjacent parts in an MPS file.

"MSK_RES_WRN_LP_OLD_QUAD_FORMAT" (80)
    Missing '/2' after quadratic expressions in bound or objective.

"MSK_RES_WRN_LP_DROP_VARIABLE" (85)
    Ignored a variable because the variable was not previously defined. Usually this implies that a variable appears in the bound section but not in the objective or the constraints.

"MSK_RES_WRN_NZ_IN_UPR_TRI" (200)
    Non-zero elements specified in the upper triangle of a matrix were ignored.

"MSK_RES_WRN_DROPPED_NZ_QOBJ" (201)
    One or more non-zero elements were dropped in the Q matrix in the objective.

"MSK_RES_WRN_IGNORE_INTEGER" (250)
    Ignored integer constraints.

"MSK_RES_WRN_NO_GLOBAL_OPTIMIZER" (251)
    No global optimizer is available.

"MSK_RES_WRN_MIO_INFEASIBLE_FINAL" (270)
    The final mixed-integer problem with all the integer variables fixed at their optimal values is infeasible.

"MSK_RES_WRN_SOL_FILTER" (300)
    Invalid solution filter is specified.

"MSK_RES_WRN_UNDEF_SOL_FILE_NAME" (350)
    Undefined name occurred in a solution.

"MSK_RES_WRN_SOL_FILE_IGNORED_CON" (351)
    One or more lines in the constraint section were ignored when reading a solution file.

"MSK_RES_WRN_SOL_FILE_IGNORED_VAR" (352)
    One or more lines in the variable section were ignored when reading a solution file.

"MSK_RES_WRN_TOO_FEW_BASIS_VARS" (400)
    An incomplete basis has been specified. Too few basis variables are specified.

"MSK_RES_WRN_TOO_MANY_BASIS_VARS" (405)
    A basis with too many variables has been specified.

"MSK_RES_WRN_LICENSE_EXPIRE" (500)
    The license expires.

"MSK_RES_WRN_LICENSE_SERVER" (501)
    The license server is not responding.

"MSK_RES_WRN_EMPTY_NAME" (502)
    A variable or constraint name is empty. The output file may be invalid.

"MSK_RES_WRN_USING_GENERIC_NAMES" (503)
    Generic names are used because a name is not valid. For instance when writing an LP file the names must not contain blanks or start with a digit.

"MSK_RES_WRN_LICENSE_FEATURE_EXPIRE" (505)
    The license expires.

"MSK_RES_WRN_PARAM_NAME_DOU" (510)
    The parameter name is not recognized as a double parameter.

"MSK_RES_WRN_PARAM_NAME_INT" (511)
The parameter name is not recognized as a integer parameter.

"MSK_RES_WRN_PARAM_NAME_STR" (512)
The parameter name is not recognized as a string parameter.

"MSK_RES_WRN_PARAM_STR_VALUE" (515)
The string is not recognized as a symbolic value for the parameter.

"MSK_RES_WRN_PARAM_IGNORED_CMIO" (516)
A parameter was ignored by the conic mixed integer optimizer.

"MSK_RES_WRN_ZEROS_IN_SPARSE_ROW" (705)
One or more (near) zero elements are specified in a sparse row of a matrix. Since, it is redundant to specify zero elements then it may indicate an error.

"MSK_RES_WRN_ZEROS_IN_SPARSE_COL" (710)
One or more (near) zero elements are specified in a sparse column of a matrix. It is redundant to specify zero elements. Hence, it may indicate an error.

"MSK_RES_WRN_INCOMPLETE_LINEAR_DEPENDENCY_CHECK" (800)
The linear dependency check(s) is incomplete. Normally this is not an important warning unless the optimization problem has been formulated with linear dependencies. Linear dependencies may prevent **MOSEK** from solving the problem.

"MSK_RES_WRN_ELIMINATOR_SPACE" (801)
The eliminator is skipped at least once due to lack of space.

"MSK_RES_WRN_PRESOLVE_OUTOFSPACE" (802)
The presolve is incomplete due to lack of space.

"MSK_RES_WRN_WRITE_CHANGED_NAMES" (803)
Some names were changed because they were invalid for the output file format.

"MSK_RES_WRN_WRITE_DISCARDED_CFIX" (804)
The fixed objective term could not be converted to a variable and was discarded in the output file.

"MSK_RES_WRN_DUPLICATE_CONSTRAINT_NAMES" (850)
Two constraint names are identical.

"MSK_RES_WRN_DUPLICATE_VARIABLE_NAMES" (851)
Two variable names are identical.

"MSK_RES_WRN_DUPLICATE_BARVARIABLE_NAMES" (852)
Two barvariable names are identical.

"MSK_RES_WRN_DUPLICATE_CONE_NAMES" (853)
Two cone names are identical.

"MSK_RES_WRN_ANA_LARGE_BOUNDS" (900)
This warning is issued by the problem analyzer, if one or more constraint or variable bounds are very large. One should consider omitting these bounds entirely by setting them to +inf or -inf.

"MSK_RES_WRN_ANA_C_ZERO" (901)
This warning is issued by the problem analyzer, if the coefficients in the linear part of the objective are all zero.

"MSK_RES_WRN_ANA_EMPTY_COLS" (902)
This warning is issued by the problem analyzer, if columns, in which all coefficients are zero, are found.

"MSK_RES_WRN_ANA_CLOSE_BOUNDS" (903)
This warning is issued by problem analyzer, if ranged constraints or variables with very close upper and lower bounds are detected. One should consider treating such constraints as equalities and such variables as constants.

"MSK_RES_WRN_ANA_ALMOST_INT_BOUNDS" (904)
This warning is issued by the problem analyzer if a constraint is bound nearly integral.

"MSK_RES_WRN_QUAD_CONES_WITH_ROOT_FIXED_AT_ZERO" (930)
For at least one quadratic cone the root is fixed at (nearly) zero. This may cause problems such as a very large dual solution. Therefore, it is recommended to remove such cones before optimizing the problem, or to fix all the variables in the cone to 0.

"MSK_RES_WRN_RQUAD_CONES_WITH_ROOT_FIXED_AT_ZERO" (931)
For at least one rotated quadratic cone at least one of the root variables are fixed at (nearly) zero. This may cause problems such as a very large dual solution. Therefore, it is recommended to remove such cones before optimizing the problem, or to fix all the variables in the cone to 0.

"MSK_RES_WRN_EXP_CONES_WITH_VARIABLES_FIXED_AT_ZERO" (932)
: For at least one exponential cone $x \geq y \exp(z/y)$ either the variable x or y is fixed at (nearly) zero. This may cause problems such as a very large dual solution. Therefore, it is recommended to remove such cones before optimizing the problem, or to fix all the variables in the cone to 0.

"MSK_RES_WRN_POW_CONES_WITH_ROOT_FIXED_AT_ZERO" (933)
: For at least one power cone at least one of the root variables are fixed at (nearly) zero. This may cause problems such as a very large dual solution. Therefore, it is recommended to remove such cones before optimizing the problem, or to fix all the variables in the cone to 0.

"MSK_RES_WRN_NO_DUALIZER" (950)
: No automatic dualizer is available for the specified problem. The primal problem is solved.

"MSK_RES_WRN_SYM_MAT_LARGE" (960)
: A numerically large value is specified for an $e_{i,j}$ element in E. The parameter *MSK_DPAR_DATA_SYM_MAT_TOL_LARGE* controls when an $e_{i,j}$ is considered large.

15.6.3 Errors

"MSK_RES_ERR_LICENSE" (1000)
: Invalid license.

"MSK_RES_ERR_LICENSE_EXPIRED" (1001)
: The license has expired.

"MSK_RES_ERR_LICENSE_VERSION" (1002)
: The license is valid for another version of **MOSEK**.

"MSK_RES_ERR_SIZE_LICENSE" (1005)
: The problem is bigger than the license.

"MSK_RES_ERR_PROB_LICENSE" (1006)
: The software is not licensed to solve the problem.

"MSK_RES_ERR_FILE_LICENSE" (1007)
: Invalid license file.

"MSK_RES_ERR_MISSING_LICENSE_FILE" (1008)
: **MOSEK** cannot find license file or a token server. See the **MOSEK** licensing manual for details.

"MSK_RES_ERR_SIZE_LICENSE_CON" (1010)
: The problem has too many constraints to be solved with the available license.

"MSK_RES_ERR_SIZE_LICENSE_VAR" (1011)
: The problem has too many variables to be solved with the available license.

"MSK_RES_ERR_SIZE_LICENSE_INTVAR" (1012)
: The problem contains too many integer variables to be solved with the available license.

"MSK_RES_ERR_OPTIMIZER_LICENSE" (1013)
: The optimizer required is not licensed.

"MSK_RES_ERR_FLEXLM" (1014)
: The FLEXlm license manager reported an error.

"MSK_RES_ERR_LICENSE_SERVER" (1015)
: The license server is not responding.

"MSK_RES_ERR_LICENSE_MAX" (1016)
: Maximum number of licenses is reached.

"MSK_RES_ERR_LICENSE_MOSEKLM_DAEMON" (1017)
: The MOSEKLM license manager daemon is not up and running.

"MSK_RES_ERR_LICENSE_FEATURE" (1018)
: A requested feature is not available in the license file(s). Most likely due to an incorrect license system setup.

"MSK_RES_ERR_PLATFORM_NOT_LICENSED" (1019)
: A requested license feature is not available for the required platform.

"MSK_RES_ERR_LICENSE_CANNOT_ALLOCATE" (1020)
: The license system cannot allocate the memory required.

"MSK_RES_ERR_LICENSE_CANNOT_CONNECT" (1021)
: **MOSEK** cannot connect to the license server. Most likely the license server is not up and running.

"MSK_RES_ERR_LICENSE_INVALID_HOSTID" (1025)
: The host ID specified in the license file does not match the host ID of the computer.

"MSK_RES_ERR_LICENSE_SERVER_VERSION" (1026)
The version specified in the checkout request is greater than the highest version number the daemon supports.

"MSK_RES_ERR_LICENSE_NO_SERVER_SUPPORT" (1027)
The license server does not support the requested feature. Possible reasons for this error include:

- The feature has expired.
- The feature's start date is later than today's date.
- The version requested is higher than feature's the highest supported version.
- A corrupted license file.

Try restarting the license and inspect the license server debug file, usually called `lmgrd.log`.

"MSK_RES_ERR_LICENSE_NO_SERVER_LINE" (1028)
There is no `SERVER` line in the license file. All non-zero license count features need at least one `SERVER` line.

"MSK_RES_ERR_OLDER_DLL" (1035)
The dynamic link library is older than the specified version.

"MSK_RES_ERR_NEWER_DLL" (1036)
The dynamic link library is newer than the specified version.

"MSK_RES_ERR_LINK_FILE_DLL" (1040)
A file cannot be linked to a stream in the DLL version.

"MSK_RES_ERR_THREAD_MUTEX_INIT" (1045)
Could not initialize a mutex.

"MSK_RES_ERR_THREAD_MUTEX_LOCK" (1046)
Could not lock a mutex.

"MSK_RES_ERR_THREAD_MUTEX_UNLOCK" (1047)
Could not unlock a mutex.

"MSK_RES_ERR_THREAD_CREATE" (1048)
Could not create a thread. This error may occur if a large number of environments are created and not deleted again. In any case it is a good practice to minimize the number of environments created.

"MSK_RES_ERR_THREAD_COND_INIT" (1049)
Could not initialize a condition.

"MSK_RES_ERR_UNKNOWN" (1050)
Unknown error.

"MSK_RES_ERR_SPACE" (1051)
Out of space.

"MSK_RES_ERR_FILE_OPEN" (1052)
Error while opening a file.

"MSK_RES_ERR_FILE_READ" (1053)
File read error.

"MSK_RES_ERR_FILE_WRITE" (1054)
File write error.

"MSK_RES_ERR_DATA_FILE_EXT" (1055)
The data file format cannot be determined from the file name.

"MSK_RES_ERR_INVALID_FILE_NAME" (1056)
An invalid file name has been specified.

"MSK_RES_ERR_INVALID_SOL_FILE_NAME" (1057)
An invalid file name has been specified.

"MSK_RES_ERR_END_OF_FILE" (1059)
End of file reached.

"MSK_RES_ERR_NULL_ENV" (1060)
`env` is a NULL pointer.

"MSK_RES_ERR_NULL_TASK" (1061)
`task` is a NULL pointer.

"MSK_RES_ERR_INVALID_STREAM" (1062)
An invalid stream is referenced.

"MSK_RES_ERR_NO_INIT_ENV" (1063)
: env is not initialized.

"MSK_RES_ERR_INVALID_TASK" (1064)
: The task is invalid.

"MSK_RES_ERR_NULL_POINTER" (1065)
: An argument to a function is unexpectedly a NULL pointer.

"MSK_RES_ERR_LIVING_TASKS" (1066)
: All tasks associated with an enviroment must be deleted before the environment is deleted. There are still some undeleted tasks.

"MSK_RES_ERR_BLANK_NAME" (1070)
: An all blank name has been specified.

"MSK_RES_ERR_DUP_NAME" (1071)
: The same name was used multiple times for the same problem item type.

"MSK_RES_ERR_FORMAT_STRING" (1072)
: The name format string is invalid.

"MSK_RES_ERR_INVALID_OBJ_NAME" (1075)
: An invalid objective name is specified.

"MSK_RES_ERR_INVALID_CON_NAME" (1076)
: An invalid constraint name is used.

"MSK_RES_ERR_INVALID_VAR_NAME" (1077)
: An invalid variable name is used.

"MSK_RES_ERR_INVALID_CONE_NAME" (1078)
: An invalid cone name is used.

"MSK_RES_ERR_INVALID_BARVAR_NAME" (1079)
: An invalid symmetric matrix variable name is used.

"MSK_RES_ERR_SPACE_LEAKING" (1080)
: **MOSEK** is leaking memory. This can be due to either an incorrect use of **MOSEK** or a bug.

"MSK_RES_ERR_SPACE_NO_INFO" (1081)
: No available information about the space usage.

"MSK_RES_ERR_READ_FORMAT" (1090)
: The specified format cannot be read.

"MSK_RES_ERR_MPS_FILE" (1100)
: An error occurred while reading an MPS file.

"MSK_RES_ERR_MPS_INV_FIELD" (1101)
: A field in the MPS file is invalid. Probably it is too wide.

"MSK_RES_ERR_MPS_INV_MARKER" (1102)
: An invalid marker has been specified in the MPS file.

"MSK_RES_ERR_MPS_NULL_CON_NAME" (1103)
: An empty constraint name is used in an MPS file.

"MSK_RES_ERR_MPS_NULL_VAR_NAME" (1104)
: An empty variable name is used in an MPS file.

"MSK_RES_ERR_MPS_UNDEF_CON_NAME" (1105)
: An undefined constraint name occurred in an MPS file.

"MSK_RES_ERR_MPS_UNDEF_VAR_NAME" (1106)
: An undefined variable name occurred in an MPS file.

"MSK_RES_ERR_MPS_INV_CON_KEY" (1107)
: An invalid constraint key occurred in an MPS file.

"MSK_RES_ERR_MPS_INV_BOUND_KEY" (1108)
: An invalid bound key occurred in an MPS file.

"MSK_RES_ERR_MPS_INV_SEC_NAME" (1109)
: An invalid section name occurred in an MPS file.

"MSK_RES_ERR_MPS_NO_OBJECTIVE" (1110)
: No objective is defined in an MPS file.

"MSK_RES_ERR_MPS_SPLITTED_VAR" (1111)
: All elements in a column of the A matrix must be specified consecutively. Hence, it is illegal to specify non-zero elements in A for variable 1, then for variable 2 and then variable 1 again.

"MSK_RES_ERR_MPS_MUL_CON_NAME" (1112)
: A constraint name was specified multiple times in the ROWS section.

"MSK_RES_ERR_MPS_MUL_QSEC" (1113)
  Multiple QSECTIONs are specified for a constraint in the MPS data file.
"MSK_RES_ERR_MPS_MUL_QOBJ" (1114)
  The Q term in the objective is specified multiple times in the MPS data file.
"MSK_RES_ERR_MPS_INV_SEC_ORDER" (1115)
  The sections in the MPS data file are not in the correct order.
"MSK_RES_ERR_MPS_MUL_CSEC" (1116)
  Multiple CSECTIONs are given the same name.
"MSK_RES_ERR_MPS_CONE_TYPE" (1117)
  Invalid cone type specified in a CSECTION.
"MSK_RES_ERR_MPS_CONE_OVERLAP" (1118)
  A variable is specified to be a member of several cones.
"MSK_RES_ERR_MPS_CONE_REPEAT" (1119)
  A variable is repeated within the CSECTION.
"MSK_RES_ERR_MPS_NON_SYMMETRIC_Q" (1120)
  A non symmetric matrix has been speciefied.
"MSK_RES_ERR_MPS_DUPLICATE_Q_ELEMENT" (1121)
  Duplicate elements is specfied in a Q matrix.
"MSK_RES_ERR_MPS_INVALID_OBJSENSE" (1122)
  An invalid objective sense is specified.
"MSK_RES_ERR_MPS_TAB_IN_FIELD2" (1125)
  A tab char occurred in field 2.
"MSK_RES_ERR_MPS_TAB_IN_FIELD3" (1126)
  A tab char occurred in field 3.
"MSK_RES_ERR_MPS_TAB_IN_FIELD5" (1127)
  A tab char occurred in field 5.
"MSK_RES_ERR_MPS_INVALID_OBJ_NAME" (1128)
  An invalid objective name is specified.
"MSK_RES_ERR_LP_INCOMPATIBLE" (1150)
  The problem cannot be written to an LP formatted file.
"MSK_RES_ERR_LP_EMPTY" (1151)
  The problem cannot be written to an LP formatted file.
"MSK_RES_ERR_LP_DUP_SLACK_NAME" (1152)
  The name of the slack variable added to a ranged constraint already exists.
"MSK_RES_ERR_WRITE_MPS_INVALID_NAME" (1153)
  An invalid name is created while writing an MPS file. Usually this will make the MPS file unreadable.
"MSK_RES_ERR_LP_INVALID_VAR_NAME" (1154)
  A variable name is invalid when used in an LP formatted file.
"MSK_RES_ERR_LP_FREE_CONSTRAINT" (1155)
  Free constraints cannot be written in LP file format.
"MSK_RES_ERR_WRITE_OPF_INVALID_VAR_NAME" (1156)
  Empty variable names cannot be written to OPF files.
"MSK_RES_ERR_LP_FILE_FORMAT" (1157)
  Syntax error in an LP file.
"MSK_RES_ERR_WRITE_LP_FORMAT" (1158)
  Problem cannot be written as an LP file.
"MSK_RES_ERR_READ_LP_MISSING_END_TAG" (1159)
  Syntax error in LP file. Possibly missing End tag.
"MSK_RES_ERR_LP_FORMAT" (1160)
  Syntax error in an LP file.
"MSK_RES_ERR_WRITE_LP_NON_UNIQUE_NAME" (1161)
  An auto-generated name is not unique.
"MSK_RES_ERR_READ_LP_NONEXISTING_NAME" (1162)
  A variable never occurred in objective or constraints.
"MSK_RES_ERR_LP_WRITE_CONIC_PROBLEM" (1163)
  The problem contains cones that cannot be written to an LP formatted file.

"MSK_RES_ERR_LP_WRITE_GECO_PROBLEM" (1164)
: The problem contains general convex terms that cannot be written to an LP formatted file.

"MSK_RES_ERR_WRITING_FILE" (1166)
: An error occurred while writing file

"MSK_RES_ERR_PTF_FORMAT" (1167)
: Syntax error in an PTF file

"MSK_RES_ERR_OPF_FORMAT" (1168)
: Syntax error in an OPF file

"MSK_RES_ERR_OPF_NEW_VARIABLE" (1169)
: Introducing new variables is now allowed. When a [variables] section is present, it is not allowed to introduce new variables later in the problem.

"MSK_RES_ERR_INVALID_NAME_IN_SOL_FILE" (1170)
: An invalid name occurred in a solution file.

"MSK_RES_ERR_LP_INVALID_CON_NAME" (1171)
: A constraint name is invalid when used in an LP formatted file.

"MSK_RES_ERR_OPF_PREMATURE_EOF" (1172)
: Premature end of file in an OPF file.

"MSK_RES_ERR_JSON_SYNTAX" (1175)
: Syntax error in an JSON data

"MSK_RES_ERR_JSON_STRING" (1176)
: Error in JSON string.

"MSK_RES_ERR_JSON_NUMBER_OVERFLOW" (1177)
: Invalid number entry - wrong type or value overflow.

"MSK_RES_ERR_JSON_FORMAT" (1178)
: Error in an JSON Task file

"MSK_RES_ERR_JSON_DATA" (1179)
: Inconsistent data in JSON Task file

"MSK_RES_ERR_JSON_MISSING_DATA" (1180)
: Missing data section in JSON task file.

"MSK_RES_ERR_ARGUMENT_LENNEQ" (1197)
: Incorrect length of arguments.

"MSK_RES_ERR_ARGUMENT_TYPE" (1198)
: Incorrect argument type.

"MSK_RES_ERR_NUM_ARGUMENTS" (1199)
: Incorrect number of function arguments.

"MSK_RES_ERR_IN_ARGUMENT" (1200)
: A function argument is incorrect.

"MSK_RES_ERR_ARGUMENT_DIMENSION" (1201)
: A function argument is of incorrect dimension.

"MSK_RES_ERR_SHAPE_IS_TOO_LARGE" (1202)
: The size of the n-dimensional shape is too large.

"MSK_RES_ERR_INDEX_IS_TOO_SMALL" (1203)
: An index in an argument is too small.

"MSK_RES_ERR_INDEX_IS_TOO_LARGE" (1204)
: An index in an argument is too large.

"MSK_RES_ERR_PARAM_NAME" (1205)
: The parameter name is not correct.

"MSK_RES_ERR_PARAM_NAME_DOU" (1206)
: The parameter name is not correct for a double parameter.

"MSK_RES_ERR_PARAM_NAME_INT" (1207)
: The parameter name is not correct for an integer parameter.

"MSK_RES_ERR_PARAM_NAME_STR" (1208)
: The parameter name is not correct for a string parameter.

"MSK_RES_ERR_PARAM_INDEX" (1210)
: Parameter index is out of range.

"MSK_RES_ERR_PARAM_IS_TOO_LARGE" (1215)
: The parameter value is too large.

"MSK_RES_ERR_PARAM_IS_TOO_SMALL" (1216)
: The parameter value is too small.

"MSK_RES_ERR_PARAM_VALUE_STR" (1217)
: The parameter value string is incorrect.

"MSK_RES_ERR_PARAM_TYPE" (1218)
: The parameter type is invalid.

"MSK_RES_ERR_INF_DOU_INDEX" (1219)
: A double information index is out of range for the specified type.

"MSK_RES_ERR_INF_INT_INDEX" (1220)
: An integer information index is out of range for the specified type.

"MSK_RES_ERR_INDEX_ARR_IS_TOO_SMALL" (1221)
: An index in an array argument is too small.

"MSK_RES_ERR_INDEX_ARR_IS_TOO_LARGE" (1222)
: An index in an array argument is too large.

"MSK_RES_ERR_INF_LINT_INDEX" (1225)
: A long integer information index is out of range for the specified type.

"MSK_RES_ERR_ARG_IS_TOO_SMALL" (1226)
: The value of a argument is too small.

"MSK_RES_ERR_ARG_IS_TOO_LARGE" (1227)
: The value of a argument is too large.

"MSK_RES_ERR_INVALID_WHICHSOL" (1228)
: `whichsol` is invalid.

"MSK_RES_ERR_INF_DOU_NAME" (1230)
: A double information name is invalid.

"MSK_RES_ERR_INF_INT_NAME" (1231)
: An integer information name is invalid.

"MSK_RES_ERR_INF_TYPE" (1232)
: The information type is invalid.

"MSK_RES_ERR_INF_LINT_NAME" (1234)
: A long integer information name is invalid.

"MSK_RES_ERR_INDEX" (1235)
: An index is out of range.

"MSK_RES_ERR_WHICHSOL" (1236)
: The solution defined by `whichsol` does not exists.

"MSK_RES_ERR_SOLITEM" (1237)
: The solution item number `solitem` is invalid. Please note that *"MSK_SOL_ITEM_SNX"* is invalid for the basic solution.

"MSK_RES_ERR_WHICHITEM_NOT_ALLOWED" (1238)
: `whichitem` is unacceptable.

"MSK_RES_ERR_MAXNUMCON" (1240)
: The maximum number of constraints specified is smaller than the number of constraints in the task.

"MSK_RES_ERR_MAXNUMVAR" (1241)
: The maximum number of variables specified is smaller than the number of variables in the task.

"MSK_RES_ERR_MAXNUMBARVAR" (1242)
: The maximum number of semidefinite variables specified is smaller than the number of semidefinite variables in the task.

"MSK_RES_ERR_MAXNUMQNZ" (1243)
: The maximum number of non-zeros specified for the Q matrices is smaller than the number of non-zeros in the current Q matrices.

"MSK_RES_ERR_TOO_SMALL_MAX_NUM_NZ" (1245)
: The maximum number of non-zeros specified is too small.

"MSK_RES_ERR_INVALID_IDX" (1246)
: A specified index is invalid.

"MSK_RES_ERR_INVALID_MAX_NUM" (1247)
: A specified index is invalid.

"MSK_RES_ERR_NUMCONLIM" (1250)
: Maximum number of constraints limit is exceeded.

"MSK_RES_ERR_NUMVARLIM" (1251)
Maximum number of variables limit is exceeded.

"MSK_RES_ERR_TOO_SMALL_MAXNUMANZ" (1252)
The maximum number of non-zeros specified for A is smaller than the number of non-zeros in the current A.

"MSK_RES_ERR_INV_APTRE" (1253)
aptre[j] is strictly smaller than aptrb[j] for some j.

"MSK_RES_ERR_MUL_A_ELEMENT" (1254)
An element in A is defined multiple times.

"MSK_RES_ERR_INV_BK" (1255)
Invalid bound key.

"MSK_RES_ERR_INV_BKC" (1256)
Invalid bound key is specified for a constraint.

"MSK_RES_ERR_INV_BKX" (1257)
An invalid bound key is specified for a variable.

"MSK_RES_ERR_INV_VAR_TYPE" (1258)
An invalid variable type is specified for a variable.

"MSK_RES_ERR_SOLVER_PROBTYPE" (1259)
Problem type does not match the chosen optimizer.

"MSK_RES_ERR_OBJECTIVE_RANGE" (1260)
Empty objective range.

"MSK_RES_ERR_UNDEF_SOLUTION" (1265)
MOSEK has the following solution types:

- an interior-point solution,
- a basic solution,
- and an integer solution.

Each optimizer may set one or more of these solutions; e.g by default a successful optimization with the interior-point optimizer defines the interior-point solution and, for linear problems, also the basic solution. This error occurs when asking for a solution or for information about a solution that is not defined.

"MSK_RES_ERR_BASIS" (1266)
An invalid basis is specified. Either too many or too few basis variables are specified.

"MSK_RES_ERR_INV_SKC" (1267)
Invalid value in skc.

"MSK_RES_ERR_INV_SKX" (1268)
Invalid value in skx.

"MSK_RES_ERR_INV_SKN" (1274)
Invalid value in skn.

"MSK_RES_ERR_INV_SK_STR" (1269)
Invalid status key string encountered.

"MSK_RES_ERR_INV_SK" (1270)
Invalid status key code.

"MSK_RES_ERR_INV_CONE_TYPE_STR" (1271)
Invalid cone type string encountered.

"MSK_RES_ERR_INV_CONE_TYPE" (1272)
Invalid cone type code is encountered.

"MSK_RES_ERR_INVALID_SURPLUS" (1275)
Invalid surplus.

"MSK_RES_ERR_INV_NAME_ITEM" (1280)
An invalid name item code is used.

"MSK_RES_ERR_PRO_ITEM" (1281)
An invalid problem is used.

"MSK_RES_ERR_INVALID_FORMAT_TYPE" (1283)
Invalid format type.

"MSK_RES_ERR_FIRSTI" (1285)
Invalid firsti.

"MSK_RES_ERR_LASTI" (1286)
Invalid lasti.

"MSK_RES_ERR_FIRSTJ" (1287)
Invalid firstj.

"MSK_RES_ERR_LASTJ" (1288)
Invalid lastj.

"MSK_RES_ERR_MAX_LEN_IS_TOO_SMALL" (1289)
A maximum length that is too small has been specified.

"MSK_RES_ERR_NONLINEAR_EQUALITY" (1290)
The model contains a nonlinear equality which defines a nonconvex set.

"MSK_RES_ERR_NONCONVEX" (1291)
The optimization problem is nonconvex.

"MSK_RES_ERR_NONLINEAR_RANGED" (1292)
Nonlinear constraints with finite lower and upper bound always define a nonconvex feasible set.

"MSK_RES_ERR_CON_Q_NOT_PSD" (1293)
The quadratic constraint matrix is not positive semidefinite as expected for a constraint with finite upper bound. This results in a nonconvex problem. The parameter *MSK_DPAR_CHECK_CONVEXITY_REL_TOL* can be used to relax the convexity check.

"MSK_RES_ERR_CON_Q_NOT_NSD" (1294)
The quadratic constraint matrix is not negative semidefinite as expected for a constraint with finite lower bound. This results in a nonconvex problem. The parameter *MSK_DPAR_CHECK_CONVEXITY_REL_TOL* can be used to relax the convexity check.

"MSK_RES_ERR_OBJ_Q_NOT_PSD" (1295)
The quadratic coefficient matrix in the objective is not positive semidefinite as expected for a minimization problem. The parameter *MSK_DPAR_CHECK_CONVEXITY_REL_TOL* can be used to relax the convexity check.

"MSK_RES_ERR_OBJ_Q_NOT_NSD" (1296)
The quadratic coefficient matrix in the objective is not negative semidefinite as expected for a maximization problem. The parameter *MSK_DPAR_CHECK_CONVEXITY_REL_TOL* can be used to relax the convexity check.

"MSK_RES_ERR_ARGUMENT_PERM_ARRAY" (1299)
An invalid permutation array is specified.

"MSK_RES_ERR_CONE_INDEX" (1300)
An index of a non-existing cone has been specified.

"MSK_RES_ERR_CONE_SIZE" (1301)
A cone with incorrect number of members is specified.

"MSK_RES_ERR_CONE_OVERLAP" (1302)
One or more of the variables in the cone to be added is already member of another cone. Now assume the variable is x_j then add a new variable say x_k and the constraint

$$x_j = x_k$$

and then let x_k be member of the cone to be appended.

"MSK_RES_ERR_CONE_REP_VAR" (1303)
A variable is included multiple times in the cone.

"MSK_RES_ERR_MAXNUMCONE" (1304)
The value specified for maxnumcone is too small.

"MSK_RES_ERR_CONE_TYPE" (1305)
Invalid cone type specified.

"MSK_RES_ERR_CONE_TYPE_STR" (1306)
Invalid cone type specified.

"MSK_RES_ERR_CONE_OVERLAP_APPEND" (1307)
The cone to be appended has one variable which is already member of another cone.

"MSK_RES_ERR_REMOVE_CONE_VARIABLE" (1310)
A variable cannot be removed because it will make a cone invalid.

"MSK_RES_ERR_APPENDING_TOO_BIG_CONE" (1311)
Trying to append a too big cone.

"MSK_RES_ERR_CONE_PARAMETER" (1320)
An invalid cone parameter.

"MSK_RES_ERR_SOL_FILE_INVALID_NUMBER" (1350)
An invalid number is specified in a solution file.

"MSK_RES_ERR_HUGE_C" (1375)
A huge value in absolute size is specified for one c_j.

"MSK_RES_ERR_HUGE_AIJ" (1380)
A numerically huge value is specified for an $a_{i,j}$ element in A. The parameter *MSK_DPAR_DATA_TOL_AIJ_HUGE* controls when an $a_{i,j}$ is considered huge.

"MSK_RES_ERR_DUPLICATE_AIJ" (1385)
An element in the A matrix is specified twice.

"MSK_RES_ERR_LOWER_BOUND_IS_A_NAN" (1390)
The lower bound specified is not a number (nan).

"MSK_RES_ERR_UPPER_BOUND_IS_A_NAN" (1391)
The upper bound specified is not a number (nan).

"MSK_RES_ERR_INFINITE_BOUND" (1400)
A numerically huge bound value is specified.

"MSK_RES_ERR_INV_QOBJ_SUBI" (1401)
Invalid value in `qosubi`.

"MSK_RES_ERR_INV_QOBJ_SUBJ" (1402)
Invalid value in `qosubj`.

"MSK_RES_ERR_INV_QOBJ_VAL" (1403)
Invalid value in `qoval`.

"MSK_RES_ERR_INV_QCON_SUBK" (1404)
Invalid value in `qcsubk`.

"MSK_RES_ERR_INV_QCON_SUBI" (1405)
Invalid value in `qcsubi`.

"MSK_RES_ERR_INV_QCON_SUBJ" (1406)
Invalid value in `qcsubj`.

"MSK_RES_ERR_INV_QCON_VAL" (1407)
Invalid value in `qcval`.

"MSK_RES_ERR_QCON_SUBI_TOO_SMALL" (1408)
Invalid value in `qcsubi`.

"MSK_RES_ERR_QCON_SUBI_TOO_LARGE" (1409)
Invalid value in `qcsubi`.

"MSK_RES_ERR_QOBJ_UPPER_TRIANGLE" (1415)
An element in the upper triangle of Q^o is specified. Only elements in the lower triangle should be specified.

"MSK_RES_ERR_QCON_UPPER_TRIANGLE" (1417)
An element in the upper triangle of a Q^k is specified. Only elements in the lower triangle should be specified.

"MSK_RES_ERR_FIXED_BOUND_VALUES" (1420)
A fixed constraint/variable has been specified using the bound keys but the numerical value of the lower and upper bound is different.

"MSK_RES_ERR_TOO_SMALL_A_TRUNCATION_VALUE" (1421)
A too small value for the A trucation value is specified.

"MSK_RES_ERR_INVALID_OBJECTIVE_SENSE" (1445)
An invalid objective sense is specified.

"MSK_RES_ERR_UNDEFINED_OBJECTIVE_SENSE" (1446)
The objective sense has not been specified before the optimization.

"MSK_RES_ERR_Y_IS_UNDEFINED" (1449)
The solution item y is undefined.

"MSK_RES_ERR_NAN_IN_DOUBLE_DATA" (1450)
An invalid floating point value was used in some double data.

"MSK_RES_ERR_NAN_IN_BLC" (1461)
l^c contains an invalid floating point value, i.e. a `NaN`.

"MSK_RES_ERR_NAN_IN_BUC" (1462)
u^c contains an invalid floating point value, i.e. a `NaN`.

"MSK_RES_ERR_NAN_IN_C" (1470)
c contains an invalid floating point value, i.e. a `NaN`.

"MSK_RES_ERR_NAN_IN_BLX" (1471)
: l^x contains an invalid floating point value, i.e. a NaN.

"MSK_RES_ERR_NAN_IN_BUX" (1472)
: u^x contains an invalid floating point value, i.e. a NaN.

"MSK_RES_ERR_INVALID_AIJ" (1473)
: $a_{i,j}$ contains an invalid floating point value, i.e. a NaN or an infinite value.

"MSK_RES_ERR_SYM_MAT_INVALID" (1480)
: A symmetric matrix contains an invalid floating point value, i.e. a NaN or an infinite value.

"MSK_RES_ERR_SYM_MAT_HUGE" (1482)
: A symmetric matrix contains a huge value in absolute size. The parameter *MSK_DPAR_DATA_SYM_MAT_TOL_HUGE* controls when an $e_{i,j}$ is considered huge.

"MSK_RES_ERR_INV_PROBLEM" (1500)
: Invalid problem type. Probably a nonconvex problem has been specified.

"MSK_RES_ERR_MIXED_CONIC_AND_NL" (1501)
: The problem contains nonlinear terms conic constraints. The requested operation cannot be applied to this type of problem.

"MSK_RES_ERR_GLOBAL_INV_CONIC_PROBLEM" (1503)
: The global optimizer can only be applied to problems without semidefinite variables.

"MSK_RES_ERR_INV_OPTIMIZER" (1550)
: An invalid optimizer has been chosen for the problem.

"MSK_RES_ERR_MIO_NO_OPTIMIZER" (1551)
: No optimizer is available for the current class of integer optimization problems.

"MSK_RES_ERR_NO_OPTIMIZER_VAR_TYPE" (1552)
: No optimizer is available for this class of optimization problems.

"MSK_RES_ERR_FINAL_SOLUTION" (1560)
: An error occurred during the solution finalization.

"MSK_RES_ERR_FIRST" (1570)
: Invalid first.

"MSK_RES_ERR_LAST" (1571)
: Invalid index last. A given index was out of expected range.

"MSK_RES_ERR_SLICE_SIZE" (1572)
: Invalid slice size specified.

"MSK_RES_ERR_NEGATIVE_SURPLUS" (1573)
: Negative surplus.

"MSK_RES_ERR_NEGATIVE_APPEND" (1578)
: Cannot append a negative number.

"MSK_RES_ERR_POSTSOLVE" (1580)
: An error occurred during the postsolve. Please contact **MOSEK** support.

"MSK_RES_ERR_OVERFLOW" (1590)
: A computation produced an overflow i.e. a very large number.

"MSK_RES_ERR_NO_BASIS_SOL" (1600)
: No basic solution is defined.

"MSK_RES_ERR_BASIS_FACTOR" (1610)
: The factorization of the basis is invalid.

"MSK_RES_ERR_BASIS_SINGULAR" (1615)
: The basis is singular and hence cannot be factored.

"MSK_RES_ERR_FACTOR" (1650)
: An error occurred while factorizing a matrix.

"MSK_RES_ERR_FEASREPAIR_CANNOT_RELAX" (1700)
: An optimization problem cannot be relaxed.

"MSK_RES_ERR_FEASREPAIR_SOLVING_RELAXED" (1701)
: The relaxed problem could not be solved to optimality. Please consult the log file for further details.

"MSK_RES_ERR_FEASREPAIR_INCONSISTENT_BOUND" (1702)
: The upper bound is less than the lower bound for a variable or a constraint. Please correct this before running the feasibility repair.

"MSK_RES_ERR_REPAIR_INVALID_PROBLEM" (1710)
: The feasibility repair does not support the specified problem type.

"MSK_RES_ERR_REPAIR_OPTIMIZATION_FAILED" (1711)
Computation the optimal relaxation failed. The cause may have been numerical problems.

"MSK_RES_ERR_NAME_MAX_LEN" (1750)
A name is longer than the buffer that is supposed to hold it.

"MSK_RES_ERR_NAME_IS_NULL" (1760)
The name buffer is a NULL pointer.

"MSK_RES_ERR_INVALID_COMPRESSION" (1800)
Invalid compression type.

"MSK_RES_ERR_INVALID_IOMODE" (1801)
Invalid io mode.

"MSK_RES_ERR_NO_PRIMAL_INFEAS_CER" (2000)
A certificate of primal infeasibility is not available.

"MSK_RES_ERR_NO_DUAL_INFEAS_CER" (2001)
A certificate of infeasibility is not available.

"MSK_RES_ERR_NO_SOLUTION_IN_CALLBACK" (2500)
The required solution is not available.

"MSK_RES_ERR_INV_MARKI" (2501)
Invalid value in marki.

"MSK_RES_ERR_INV_MARKJ" (2502)
Invalid value in markj.

"MSK_RES_ERR_INV_NUMI" (2503)
Invalid numi.

"MSK_RES_ERR_INV_NUMJ" (2504)
Invalid numj.

"MSK_RES_ERR_TASK_INCOMPATIBLE" (2560)
The Task file is incompatible with this platform. This results from reading a file on a 32 bit platform generated on a 64 bit platform.

"MSK_RES_ERR_TASK_INVALID" (2561)
The Task file is invalid.

"MSK_RES_ERR_TASK_WRITE" (2562)
Failed to write the task file.

"MSK_RES_ERR_LU_MAX_NUM_TRIES" (2800)
Could not compute the LU factors of the matrix within the maximum number of allowed tries.

"MSK_RES_ERR_INVALID_UTF8" (2900)
An invalid UTF8 string is encountered.

"MSK_RES_ERR_INVALID_WCHAR" (2901)
An invalid `wchar` string is encountered.

"MSK_RES_ERR_NO_DUAL_FOR_ITG_SOL" (2950)
No dual information is available for the integer solution.

"MSK_RES_ERR_NO_SNX_FOR_BAS_SOL" (2953)
s_n^x is not available for the basis solution.

"MSK_RES_ERR_INTERNAL" (3000)
An internal error occurred. Please report this problem.

"MSK_RES_ERR_API_ARRAY_TOO_SMALL" (3001)
An input array was too short.

"MSK_RES_ERR_API_CB_CONNECT" (3002)
Failed to connect a callback object.

"MSK_RES_ERR_API_FATAL_ERROR" (3005)
An internal error occurred in the API. Please report this problem.

"MSK_RES_ERR_API_INTERNAL" (3999)
An internal fatal error occurred in an interface function.

"MSK_RES_ERR_SEN_FORMAT" (3050)
Syntax error in sensitivity analysis file.

"MSK_RES_ERR_SEN_UNDEF_NAME" (3051)
An undefined name was encountered in the sensitivity analysis file.

"MSK_RES_ERR_SEN_INDEX_RANGE" (3052)
Index out of range in the sensitivity analysis file.

"MSK_RES_ERR_SEN_BOUND_INVALID_UP" (3053)
: Analysis of upper bound requested for an index, where no upper bound exists.

"MSK_RES_ERR_SEN_BOUND_INVALID_LO" (3054)
: Analysis of lower bound requested for an index, where no lower bound exists.

"MSK_RES_ERR_SEN_INDEX_INVALID" (3055)
: Invalid range given in the sensitivity file.

"MSK_RES_ERR_SEN_INVALID_REGEXP" (3056)
: Syntax error in regexp or regexp longer than 1024.

"MSK_RES_ERR_SEN_SOLUTION_STATUS" (3057)
: No optimal solution found to the original problem given for sensitivity analysis.

"MSK_RES_ERR_SEN_NUMERICAL" (3058)
: Numerical difficulties encountered performing the sensitivity analysis.

"MSK_RES_ERR_SEN_UNHANDLED_PROBLEM_TYPE" (3080)
: Sensitivity analysis cannot be performed for the specified problem. Sensitivity analysis is only possible for linear problems.

"MSK_RES_ERR_UNB_STEP_SIZE" (3100)
: A step size in an optimizer was unexpectedly unbounded. For instance, if the step-size becomes unbounded in phase 1 of the simplex algorithm then an error occurs. Normally this will happen only if the problem is badly formulated. Please contact **MOSEK** support if this error occurs.

"MSK_RES_ERR_IDENTICAL_TASKS" (3101)
: Some tasks related to this function call were identical. Unique tasks were expected.

"MSK_RES_ERR_AD_INVALID_CODELIST" (3102)
: The code list data was invalid.

"MSK_RES_ERR_INTERNAL_TEST_FAILED" (3500)
: An internal unit test function failed.

"MSK_RES_ERR_XML_INVALID_PROBLEM_TYPE" (3600)
: The problem type is not supported by the XML format.

"MSK_RES_ERR_INVALID_AMPL_STUB" (3700)
: Invalid AMPL stub.

"MSK_RES_ERR_INT64_TO_INT32_CAST" (3800)
: A 64 bit integer could not be cast to a 32 bit integer.

"MSK_RES_ERR_SIZE_LICENSE_NUMCORES" (3900)
: The computer contains more cpu cores than the license allows for.

"MSK_RES_ERR_INFEAS_UNDEFINED" (3910)
: The requested value is not defined for this solution type.

"MSK_RES_ERR_NO_BARX_FOR_SOLUTION" (3915)
: There is no $\overline{X}$ available for the solution specified. In particular note there are no $\overline{X}$ defined for the basic and integer solutions.

"MSK_RES_ERR_NO_BARS_FOR_SOLUTION" (3916)
: There is no $\bar{s}$ available for the solution specified. In particular note there are no $\bar{s}$ defined for the basic and integer solutions.

"MSK_RES_ERR_BAR_VAR_DIM" (3920)
: The dimension of a symmetric matrix variable has to be greater than 0.

"MSK_RES_ERR_SYM_MAT_INVALID_ROW_INDEX" (3940)
: A row index specified for sparse symmetric matrix is invalid.

"MSK_RES_ERR_SYM_MAT_INVALID_COL_INDEX" (3941)
: A column index specified for sparse symmetric matrix is invalid.

"MSK_RES_ERR_SYM_MAT_NOT_LOWER_TRINGULAR" (3942)
: Only the lower triangular part of sparse symmetric matrix should be specified.

"MSK_RES_ERR_SYM_MAT_INVALID_VALUE" (3943)
: The numerical value specified in a sparse symmetric matrix is not a floating point value.

"MSK_RES_ERR_SYM_MAT_DUPLICATE" (3944)
: A value in a symmetric matric as been specified more than once.

"MSK_RES_ERR_INVALID_SYM_MAT_DIM" (3950)
: A sparse symmetric matrix of invalid dimension is specified.

"MSK_RES_ERR_INVALID_FILE_FORMAT_FOR_SYM_MAT" (4000)
: The file format does not support a problem with symmetric matrix variables.

"MSK_RES_ERR_INVALID_FILE_FORMAT_FOR_CFIX" (4001)
The file format does not support a problem with nonzero fixed term in c.
"MSK_RES_ERR_INVALID_FILE_FORMAT_FOR_RANGED_CONSTRAINTS" (4002)
The file format does not support a problem with ranged constraints.
"MSK_RES_ERR_INVALID_FILE_FORMAT_FOR_FREE_CONSTRAINTS" (4003)
The file format does not support a problem with free constraints.
"MSK_RES_ERR_INVALID_FILE_FORMAT_FOR_CONES" (4005)
The file format does not support a problem with conic constraints.
"MSK_RES_ERR_INVALID_FILE_FORMAT_FOR_NONLINEAR" (4010)
The file format does not support a problem with nonlinear terms.
"MSK_RES_ERR_DUPLICATE_CONSTRAINT_NAMES" (4500)
Two constraint names are identical.
"MSK_RES_ERR_DUPLICATE_VARIABLE_NAMES" (4501)
Two variable names are identical.
"MSK_RES_ERR_DUPLICATE_BARVARIABLE_NAMES" (4502)
Two barvariable names are identical.
"MSK_RES_ERR_DUPLICATE_CONE_NAMES" (4503)
Two cone names are identical.
"MSK_RES_ERR_NON_UNIQUE_ARRAY" (5000)
An array does not contain unique elements.
"MSK_RES_ERR_ARGUMENT_IS_TOO_LARGE" (5005)
The value of a function argument is too large.
"MSK_RES_ERR_MIO_INTERNAL" (5010)
A fatal error occurred in the mixed integer optimizer. Please contact **MOSEK** support.
"MSK_RES_ERR_INVALID_PROBLEM_TYPE" (6000)
An invalid problem type.
"MSK_RES_ERR_UNHANDLED_SOLUTION_STATUS" (6010)
Unhandled solution status.
"MSK_RES_ERR_UPPER_TRIANGLE" (6020)
An element in the upper triangle of a lower triangular matrix is specified.
"MSK_RES_ERR_LAU_SINGULAR_MATRIX" (7000)
A matrix is singular.
"MSK_RES_ERR_LAU_NOT_POSITIVE_DEFINITE" (7001)
A matrix is not positive definite.
"MSK_RES_ERR_LAU_INVALID_LOWER_TRIANGULAR_MATRIX" (7002)
An invalid lower triangular matrix.
"MSK_RES_ERR_LAU_UNKNOWN" (7005)
An unknown error.
"MSK_RES_ERR_LAU_ARG_M" (7010)
Invalid argument m.
"MSK_RES_ERR_LAU_ARG_N" (7011)
Invalid argument n.
"MSK_RES_ERR_LAU_ARG_K" (7012)
Invalid argument k.
"MSK_RES_ERR_LAU_ARG_TRANSA" (7015)
Invalid argument transa.
"MSK_RES_ERR_LAU_ARG_TRANSB" (7016)
Invalid argument transb.
"MSK_RES_ERR_LAU_ARG_UPLO" (7017)
Invalid argument uplo.
"MSK_RES_ERR_LAU_ARG_TRANS" (7018)
Invalid argument trans.
"MSK_RES_ERR_LAU_INVALID_SPARSE_SYMMETRIC_MATRIX" (7019)
An invalid sparse symmetric matrix is specfified. Note only the lower triangular part with no duplicates is specifed.
"MSK_RES_ERR_CBF_PARSE" (7100)
An error occurred while parsing an CBF file.

"MSK_RES_ERR_CBF_OBJ_SENSE" (7101)
An invalid objective sense is specified.

"MSK_RES_ERR_CBF_NO_VARIABLES" (7102)
No variables are specified.

"MSK_RES_ERR_CBF_TOO_MANY_CONSTRAINTS" (7103)
Too many constraints specified.

"MSK_RES_ERR_CBF_TOO_MANY_VARIABLES" (7104)
Too many variables specified.

"MSK_RES_ERR_CBF_NO_VERSION_SPECIFIED" (7105)
No version specified.

"MSK_RES_ERR_CBF_SYNTAX" (7106)
Invalid syntax.

"MSK_RES_ERR_CBF_DUPLICATE_OBJ" (7107)
Duplicate OBJ keyword.

"MSK_RES_ERR_CBF_DUPLICATE_CON" (7108)
Duplicate CON keyword.

"MSK_RES_ERR_CBF_DUPLICATE_VAR" (7109)
Duplicate VAR keyword.

"MSK_RES_ERR_CBF_DUPLICATE_INT" (7110)
Duplicate INT keyword.

"MSK_RES_ERR_CBF_INVALID_VAR_TYPE" (7111)
Invalid variable type.

"MSK_RES_ERR_CBF_INVALID_CON_TYPE" (7112)
Invalid constraint type.

"MSK_RES_ERR_CBF_INVALID_DOMAIN_DIMENSION" (7113)
Invalid domain dimension.

"MSK_RES_ERR_CBF_DUPLICATE_OBJACOORD" (7114)
Duplicate index in OBJCOORD.

"MSK_RES_ERR_CBF_DUPLICATE_BCOORD" (7115)
Duplicate index in BCOORD.

"MSK_RES_ERR_CBF_DUPLICATE_ACOORD" (7116)
Duplicate index in ACOORD.

"MSK_RES_ERR_CBF_TOO_FEW_VARIABLES" (7117)
Too few variables defined.

"MSK_RES_ERR_CBF_TOO_FEW_CONSTRAINTS" (7118)
Too few constraints defined.

"MSK_RES_ERR_CBF_TOO_FEW_INTS" (7119)
Too few ints are specified.

"MSK_RES_ERR_CBF_TOO_MANY_INTS" (7120)
Too many ints are specified.

"MSK_RES_ERR_CBF_INVALID_INT_INDEX" (7121)
Invalid INT index.

"MSK_RES_ERR_CBF_UNSUPPORTED" (7122)
Unsupported feature is present.

"MSK_RES_ERR_CBF_DUPLICATE_PSDVAR" (7123)
Duplicate PSDVAR keyword.

"MSK_RES_ERR_CBF_INVALID_PSDVAR_DIMENSION" (7124)
Invalid PSDVAR dimension.

"MSK_RES_ERR_CBF_TOO_FEW_PSDVAR" (7125)
Too few variables defined.

"MSK_RES_ERR_CBF_INVALID_EXP_DIMENSION" (7126)
Invalid dimension of a exponential cone.

"MSK_RES_ERR_CBF_DUPLICATE_POW_CONES" (7130)
Multiple POWCONES specified.

"MSK_RES_ERR_CBF_DUPLICATE_POW_STAR_CONES" (7131)
Multiple POW*CONES specified.

"MSK_RES_ERR_CBF_INVALID_POWER" (7132)
Invalid power specified.

"MSK_RES_ERR_CBF_POWER_CONE_IS_TOO_LONG" (7133)
Power cone is too long.

"MSK_RES_ERR_CBF_INVALID_POWER_CONE_INDEX" (7134)
Invalid power cone index.

"MSK_RES_ERR_CBF_INVALID_POWER_STAR_CONE_INDEX" (7135)
Invalid power star cone index.

"MSK_RES_ERR_CBF_UNHANDLED_POWER_CONE_TYPE" (7136)
An unhandled power cone type.

"MSK_RES_ERR_CBF_UNHANDLED_POWER_STAR_CONE_TYPE" (7137)
An unhandled power star cone type.

"MSK_RES_ERR_CBF_POWER_CONE_MISMATCH" (7138)
The power cone does not match with it definition.

"MSK_RES_ERR_CBF_POWER_STAR_CONE_MISMATCH" (7139)
The power star cone does not match with it definition.

"MSK_RES_ERR_CBF_INVALID_NUMBER_OF_CONES" (7740)
Invalid number of cones.

"MSK_RES_ERR_CBF_INVALID_DIMENSION_OF_CONES" (7741)
Invalid dimension of cones.

"MSK_RES_ERR_MIO_INVALID_ROOT_OPTIMIZER" (7700)
An invalid root optimizer was selected for the problem type.

"MSK_RES_ERR_MIO_INVALID_NODE_OPTIMIZER" (7701)
An invalid node optimizer was selected for the problem type.

"MSK_RES_ERR_TOCONIC_CONSTR_Q_NOT_PSD" (7800)
The matrix defining the quadratric part of constraint is not positive semidefinite.

"MSK_RES_ERR_TOCONIC_CONSTRAINT_FX" (7801)
The quadratic constraint is an equality, thus not convex.

"MSK_RES_ERR_TOCONIC_CONSTRAINT_RA" (7802)
The quadratic constraint has finite lower and upper bound, and therefore it is not convex.

"MSK_RES_ERR_TOCONIC_CONSTR_NOT_CONIC" (7803)
The constraint is not conic representable.

"MSK_RES_ERR_TOCONIC_OBJECTIVE_NOT_PSD" (7804)
The matrix defining the quadratric part of the objective function is not positive semidefinite.

"MSK_RES_ERR_SERVER_CONNECT" (8000)
Failed to connect to remote solver server. The server string or the port string were invalid, or the server did not accept connection.

"MSK_RES_ERR_SERVER_PROTOCOL" (8001)
Unexpected message or data from solver server.

"MSK_RES_ERR_SERVER_STATUS" (8002)
Server returned non-ok HTTP status code

"MSK_RES_ERR_SERVER_TOKEN" (8003)
The job ID specified is incorrect or invalid

15.7 Enumerations

basindtype
Basis identification

"MSK_BI_NEVER"
Never do basis identification.

"MSK_BI_ALWAYS"
Basis identification is always performed even if the interior-point optimizer terminates abnormally.

"MSK_BI_NO_ERROR"
Basis identification is performed if the interior-point optimizer terminates without an error.

"MSK_BI_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.

"MSK_BI_RESERVERED"

Not currently in use.

`boundkey`

Bound keys

"MSK_BK_LO"

The constraint or variable has a finite lower bound and an infinite upper bound.

"MSK_BK_UP"

The constraint or variable has an infinite lower bound and an finite upper bound.

"MSK_BK_FX"

The constraint or variable is fixed.

"MSK_BK_FR"

The constraint or variable is free.

"MSK_BK_RA"

The constraint or variable is ranged.

`mark`

Mark

"MSK_MARK_LO"

The lower bound is selected for sensitivity analysis.

"MSK_MARK_UP"

The upper bound is selected for sensitivity analysis.

`simdegen`

Degeneracy strategies

"MSK_SIM_DEGEN_NONE"

The simplex optimizer should use no degeneration strategy.

"MSK_SIM_DEGEN_FREE"

The simplex optimizer chooses the degeneration strategy.

"MSK_SIM_DEGEN_AGGRESSIVE"

The simplex optimizer should use an aggressive degeneration strategy.

"MSK_SIM_DEGEN_MODERATE"

The simplex optimizer should use a moderate degeneration strategy.

"MSK_SIM_DEGEN_MINIMUM"

The simplex optimizer should use a minimum degeneration strategy.

`transpose`

Transposed matrix.

"MSK_TRANSPOSE_NO"

No transpose is applied.

"MSK_TRANSPOSE_YES"

A transpose is applied.

`uplo`

Triangular part of a symmetric matrix.

"MSK_UPLO_LO"

Lower part.

"MSK_UPLO_UP"

Upper part.

`simreform`

Problem reformulation.

"MSK_SIM_REFORMULATION_ON"

Allow the simplex optimizer to reformulate the problem.

"MSK_SIM_REFORMULATION_OFF"

Disallow the simplex optimizer to reformulate the problem.

"MSK_SIM_REFORMULATION_FREE"

The simplex optimizer can choose freely.

"MSK_SIM_REFORMULATION_AGGRESSIVE"
The simplex optimizer should use an aggressive reformulation strategy.

simdupvec
Exploit duplicate columns.

"MSK_SIM_EXPLOIT_DUPVEC_ON"
Allow the simplex optimizer to exploit duplicated columns.

"MSK_SIM_EXPLOIT_DUPVEC_OFF"
Disallow the simplex optimizer to exploit duplicated columns.

"MSK_SIM_EXPLOIT_DUPVEC_FREE"
The simplex optimizer can choose freely.

simhotstart
Hot-start type employed by the simplex optimizer

"MSK_SIM_HOTSTART_NONE"
The simplex optimizer performs a coldstart.

"MSK_SIM_HOTSTART_FREE"
The simplex optimize chooses the hot-start type.

"MSK_SIM_HOTSTART_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.

"MSK_INTPNT_HOTSTART_NONE"
The interior-point optimizer performs a coldstart.

"MSK_INTPNT_HOTSTART_PRIMAL"
The interior-point optimizer exploits the primal solution only.

"MSK_INTPNT_HOTSTART_DUAL"
The interior-point optimizer exploits the dual solution only.

"MSK_INTPNT_HOTSTART_PRIMAL_DUAL"
The interior-point optimizer exploits both the primal and dual solution.

purify
Solution purification employed optimizer.

"MSK_PURIFY_NONE"
The optimizer performs no solution purification.

"MSK_PURIFY_PRIMAL"
The optimizer purifies the primal solution.

"MSK_PURIFY_DUAL"
The optimizer purifies the dual solution.

"MSK_PURIFY_PRIMAL_DUAL"
The optimizer purifies both the primal and dual solution.

"MSK_PURIFY_AUTO"
TBD

callbackcode
Progress callback codes

"MSK_CALLBACK_BEGIN_BI"
The basis identification procedure has been started.

"MSK_CALLBACK_BEGIN_CONIC"
The callback function is called when the conic optimizer is started.

"MSK_CALLBACK_BEGIN_DUAL_BI"
The callback function is called from within the basis identification procedure when the dual phase is started.

"MSK_CALLBACK_BEGIN_DUAL_SENSITIVITY"
Dual sensitivity analysis is started.

"MSK_CALLBACK_BEGIN_DUAL_SETUP_BI"
The callback function is called when the dual BI phase is started.

"MSK_CALLBACK_BEGIN_DUAL_SIMPLEX"
The callback function is called when the dual simplex optimizer started.

"MSK_CALLBACK_BEGIN_DUAL_SIMPLEX_BI"
The callback function is called from within the basis identification procedure when the dual simplex clean-up phase is started.

"MSK_CALLBACK_BEGIN_FULL_CONVEXITY_CHECK"
Begin full convexity check.

"MSK_CALLBACK_BEGIN_INFEAS_ANA"
The callback function is called when the infeasibility analyzer is started.

"MSK_CALLBACK_BEGIN_INTPNT"
The callback function is called when the interior-point optimizer is started.

"MSK_CALLBACK_BEGIN_LICENSE_WAIT"
Begin waiting for license.

"MSK_CALLBACK_BEGIN_MIO"
The callback function is called when the mixed-integer optimizer is started.

"MSK_CALLBACK_BEGIN_OPTIMIZER"
The callback function is called when the optimizer is started.

"MSK_CALLBACK_BEGIN_PRESOLVE"
The callback function is called when the presolve is started.

"MSK_CALLBACK_BEGIN_PRIMAL_BI"
The callback function is called from within the basis identification procedure when the primal phase is started.

"MSK_CALLBACK_BEGIN_PRIMAL_REPAIR"
Begin primal feasibility repair.

"MSK_CALLBACK_BEGIN_PRIMAL_SENSITIVITY"
Primal sensitivity analysis is started.

"MSK_CALLBACK_BEGIN_PRIMAL_SETUP_BI"
The callback function is called when the primal BI setup is started.

"MSK_CALLBACK_BEGIN_PRIMAL_SIMPLEX"
The callback function is called when the primal simplex optimizer is started.

"MSK_CALLBACK_BEGIN_PRIMAL_SIMPLEX_BI"
The callback function is called from within the basis identification procedure when the primal simplex clean-up phase is started.

"MSK_CALLBACK_BEGIN_QCQO_REFORMULATE"
Begin QCQO reformulation.

"MSK_CALLBACK_BEGIN_READ"
MOSEK has started reading a problem file.

"MSK_CALLBACK_BEGIN_ROOT_CUTGEN"
The callback function is called when root cut generation is started.

"MSK_CALLBACK_BEGIN_SIMPLEX"
The callback function is called when the simplex optimizer is started.

"MSK_CALLBACK_BEGIN_SIMPLEX_BI"
The callback function is called from within the basis identification procedure when the simplex clean-up phase is started.

"MSK_CALLBACK_BEGIN_TO_CONIC"
Begin conic reformulation.

"MSK_CALLBACK_BEGIN_WRITE"
MOSEK has started writing a problem file.

"MSK_CALLBACK_CONIC"
: The callback function is called from within the conic optimizer after the information database has been updated.

"MSK_CALLBACK_DUAL_SIMPLEX"
: The callback function is called from within the dual simplex optimizer.

"MSK_CALLBACK_END_BI"
: The callback function is called when the basis identification procedure is terminated.

"MSK_CALLBACK_END_CONIC"
: The callback function is called when the conic optimizer is terminated.

"MSK_CALLBACK_END_DUAL_BI"
: The callback function is called from within the basis identification procedure when the dual phase is terminated.

"MSK_CALLBACK_END_DUAL_SENSITIVITY"
: Dual sensitivity analysis is terminated.

"MSK_CALLBACK_END_DUAL_SETUP_BI"
: The callback function is called when the dual BI phase is terminated.

"MSK_CALLBACK_END_DUAL_SIMPLEX"
: The callback function is called when the dual simplex optimizer is terminated.

"MSK_CALLBACK_END_DUAL_SIMPLEX_BI"
: The callback function is called from within the basis identification procedure when the dual clean-up phase is terminated.

"MSK_CALLBACK_END_FULL_CONVEXITY_CHECK"
: End full convexity check.

"MSK_CALLBACK_END_INFEAS_ANA"
: The callback function is called when the infeasibility analyzer is terminated.

"MSK_CALLBACK_END_INTPNT"
: The callback function is called when the interior-point optimizer is terminated.

"MSK_CALLBACK_END_LICENSE_WAIT"
: End waiting for license.

"MSK_CALLBACK_END_MIO"
: The callback function is called when the mixed-integer optimizer is terminated.

"MSK_CALLBACK_END_OPTIMIZER"
: The callback function is called when the optimizer is terminated.

"MSK_CALLBACK_END_PRESOLVE"
: The callback function is called when the presolve is completed.

"MSK_CALLBACK_END_PRIMAL_BI"
: The callback function is called from within the basis identification procedure when the primal phase is terminated.

"MSK_CALLBACK_END_PRIMAL_REPAIR"
: End primal feasibility repair.

"MSK_CALLBACK_END_PRIMAL_SENSITIVITY"
: Primal sensitivity analysis is terminated.

"MSK_CALLBACK_END_PRIMAL_SETUP_BI"
: The callback function is called when the primal BI setup is terminated.

"MSK_CALLBACK_END_PRIMAL_SIMPLEX"
: The callback function is called when the primal simplex optimizer is terminated.

"MSK_CALLBACK_END_PRIMAL_SIMPLEX_BI"
: The callback function is called from within the basis identification procedure when the primal clean-up phase is terminated.

"MSK_CALLBACK_END_QCQO_REFORMULATE"
: End QCQO reformulation.

"MSK_CALLBACK_END_READ"
: **MOSEK** has finished reading a problem file.

"MSK_CALLBACK_END_ROOT_CUTGEN"
: The callback function is called when root cut generation is terminated.

"MSK_CALLBACK_END_SIMPLEX"
: The callback function is called when the simplex optimizer is terminated.

"MSK_CALLBACK_END_SIMPLEX_BI"
: The callback function is called from within the basis identification procedure when the simplex clean-up phase is terminated.

"MSK_CALLBACK_END_TO_CONIC"
: End conic reformulation.

"MSK_CALLBACK_END_WRITE"
: **MOSEK** has finished writing a problem file.

"MSK_CALLBACK_IM_BI"
: The callback function is called from within the basis identification procedure at an intermediate point.

"MSK_CALLBACK_IM_CONIC"
: The callback function is called at an intermediate stage within the conic optimizer where the information database has not been updated.

"MSK_CALLBACK_IM_DUAL_BI"
: The callback function is called from within the basis identification procedure at an intermediate point in the dual phase.

"MSK_CALLBACK_IM_DUAL_SENSIVITY"
: The callback function is called at an intermediate stage of the dual sensitivity analysis.

"MSK_CALLBACK_IM_DUAL_SIMPLEX"
: The callback function is called at an intermediate point in the dual simplex optimizer.

"MSK_CALLBACK_IM_FULL_CONVEXITY_CHECK"
: The callback function is called at an intermediate stage of the full convexity check.

"MSK_CALLBACK_IM_INTPNT"
: The callback function is called at an intermediate stage within the interior-point optimizer where the information database has not been updated.

"MSK_CALLBACK_IM_LICENSE_WAIT"
: **MOSEK** is waiting for a license.

"MSK_CALLBACK_IM_LU"
: The callback function is called from within the LU factorization procedure at an intermediate point.

"MSK_CALLBACK_IM_MIO"
: The callback function is called at an intermediate point in the mixed-integer optimizer.

"MSK_CALLBACK_IM_MIO_DUAL_SIMPLEX"
: The callback function is called at an intermediate point in the mixed-integer optimizer while running the dual simplex optimizer.

"MSK_CALLBACK_IM_MIO_INTPNT"
: The callback function is called at an intermediate point in the mixed-integer optimizer while running the interior-point optimizer.

"MSK_CALLBACK_IM_MIO_PRIMAL_SIMPLEX"
: The callback function is called at an intermediate point in the mixed-integer optimizer while running the primal simplex optimizer.

"MSK_CALLBACK_IM_ORDER"
: The callback function is called from within the matrix ordering procedure at an intermediate point.

"MSK_CALLBACK_IM_PRESOLVE"
: The callback function is called from within the presolve procedure at an intermediate stage.

"MSK_CALLBACK_IM_PRIMAL_BI"
: The callback function is called from within the basis identification procedure at an intermediate point in the primal phase.

"MSK_CALLBACK_IM_PRIMAL_SENSIVITY"
: The callback function is called at an intermediate stage of the primal sensitivity analysis.

"MSK_CALLBACK_IM_PRIMAL_SIMPLEX"
: The callback function is called at an intermediate point in the primal simplex optimizer.

"MSK_CALLBACK_IM_QO_REFORMULATE"
: The callback function is called at an intermediate stage of the conic quadratic reformulation.

"MSK_CALLBACK_IM_READ"
: Intermediate stage in reading.

"MSK_CALLBACK_IM_ROOT_CUTGEN"
: The callback is called from within root cut generation at an intermediate stage.

"MSK_CALLBACK_IM_SIMPLEX"
: The callback function is called from within the simplex optimizer at an intermediate point.

"MSK_CALLBACK_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 *MSK_IPAR_LOG_SIM_FREQ* parameter.

"MSK_CALLBACK_INTPNT"
: The callback function is called from within the interior-point optimizer after the information database has been updated.

"MSK_CALLBACK_NEW_INT_MIO"
: The callback function is called after a new integer solution has been located by the mixed-integer optimizer.

"MSK_CALLBACK_PRIMAL_SIMPLEX"
: The callback function is called from within the primal simplex optimizer.

"MSK_CALLBACK_READ_OPF"
: The callback function is called from the OPF reader.

"MSK_CALLBACK_READ_OPF_SECTION"
: A chunk of Q non-zeros has been read from a problem file.

"MSK_CALLBACK_SOLVING_REMOTE"
: The callback function is called while the task is being solved on a remote server.

"MSK_CALLBACK_UPDATE_DUAL_BI"
: The callback function is called from within the basis identification procedure at an intermediate point in the dual phase.

"MSK_CALLBACK_UPDATE_DUAL_SIMPLEX"
: The callback function is called in the dual simplex optimizer.

"MSK_CALLBACK_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 *MSK_IPAR_LOG_SIM_FREQ* parameter.

"MSK_CALLBACK_UPDATE_PRESOLVE"
: The callback function is called from within the presolve procedure.

"MSK_CALLBACK_UPDATE_PRIMAL_BI"
: The callback function is called from within the basis identification procedure at an intermediate point in the primal phase.

"MSK_CALLBACK_UPDATE_PRIMAL_SIMPLEX"
: The callback function is called in the primal simplex optimizer.

"MSK_CALLBACK_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 *MSK_IPAR_LOG_SIM_FREQ* parameter.

"MSK_CALLBACK_WRITE_OPF"

The callback function is called from the OPF writer.

`checkconvexitytype`

Types of convexity checks.

"MSK_CHECK_CONVEXITY_NONE"

No convexity check.

"MSK_CHECK_CONVEXITY_SIMPLE"

Perform simple and fast convexity check.

"MSK_CHECK_CONVEXITY_FULL"

Perform a full convexity check.

`compresstype`

Compression types

"MSK_COMPRESS_NONE"

No compression is used.

"MSK_COMPRESS_FREE"

The type of compression used is chosen automatically.

"MSK_COMPRESS_GZIP"

The type of compression used is gzip compatible.

"MSK_COMPRESS_ZSTD"

The type of compression used is zstd compatible.

`conetype`

Cone types

"MSK_CT_QUAD"

The cone is a quadratic cone.

"MSK_CT_RQUAD"

The cone is a rotated quadratic cone.

"MSK_CT_PEXP"

A primal exponential cone.

"MSK_CT_DEXP"

A dual exponential cone.

"MSK_CT_PPOW"

A primal power cone.

"MSK_CT_DPOW"

A dual power cone.

"MSK_CT_ZERO"

The zero cone.

`nametype`

Name types

"MSK_NAME_TYPE_GEN"

General names. However, no duplicate and blank names are allowed.

"MSK_NAME_TYPE_MPS"

MPS type names.

"MSK_NAME_TYPE_LP"

LP type names.

`scopr`

SCopt operator types

"MSK_OPR_ENT"

Entropy

"MSK_OPR_EXP"

Exponential

"MSK_OPR_LOG"

Logarithm

"MSK_OPR_POW"
: Power

"MSK_OPR_SQRT"
: Square root

symmattype
: Cone types

"MSK_SYMMAT_TYPE_SPARSE"
: Sparse symmetric matrix.

dataformat
: Data format types

"MSK_DATA_FORMAT_EXTENSION"
: The file extension is used to determine the data file format.

"MSK_DATA_FORMAT_MPS"
: The data file is MPS formatted.

"MSK_DATA_FORMAT_LP"
: The data file is LP formatted.

"MSK_DATA_FORMAT_OP"
: The data file is an optimization problem formatted file.

"MSK_DATA_FORMAT_FREE_MPS"
: The data a free MPS formatted file.

"MSK_DATA_FORMAT_TASK"
: Generic task dump file.

"MSK_DATA_FORMAT_PTF"
: (P)retty (T)ext (F)format.

"MSK_DATA_FORMAT_CB"
: Conic benchmark format,

"MSK_DATA_FORMAT_JSON_TASK"
: JSON based task format.

dinfitem
: Double information items

"MSK_DINF_BI_CLEAN_DUAL_TIME"
: Time spent within the dual clean-up optimizer of the basis identification procedure since its invocation.

"MSK_DINF_BI_CLEAN_PRIMAL_TIME"
: Time spent within the primal clean-up optimizer of the basis identification procedure since its invocation.

"MSK_DINF_BI_CLEAN_TIME"
: Time spent within the clean-up phase of the basis identification procedure since its invocation.

"MSK_DINF_BI_DUAL_TIME"
: Time spent within the dual phase basis identification procedure since its invocation.

"MSK_DINF_BI_PRIMAL_TIME"
: Time spent within the primal phase of the basis identification procedure since its invocation.

"MSK_DINF_BI_TIME"
: Time spent within the basis identification procedure since its invocation.

"MSK_DINF_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.)

"MSK_DINF_INTPNT_DUAL_OBJ"
: Dual objective value reported by the interior-point optimizer.

"MSK_DINF_INTPNT_FACTOR_NUM_FLOPS"
: An estimate of the number of flops used in the factorization.

"MSK_DINF_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.

"MSK_DINF_INTPNT_ORDER_TIME"
: Order time (in seconds).

"MSK_DINF_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).

"MSK_DINF_INTPNT_PRIMAL_OBJ"
: Primal objective value reported by the interior-point optimizer.

"MSK_DINF_INTPNT_TIME"
: Time spent within the interior-point optimizer since its invocation.

"MSK_DINF_MIO_CLIQUE_SEPARATION_TIME"
: Separation time for clique cuts.

"MSK_DINF_MIO_CMIR_SEPARATION_TIME"
: Separation time for CMIR cuts.

"MSK_DINF_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.

"MSK_DINF_MIO_DUAL_BOUND_AFTER_PRESOLVE"
: Value of the dual bound after presolve but before cut generation.

"MSK_DINF_MIO_GMI_SEPARATION_TIME"
: Separation time for GMI cuts.

"MSK_DINF_MIO_IMPLIED_BOUND_TIME"
: Separation time for implied bound cuts.

"MSK_DINF_MIO_KNAPSACK_COVER_SEPARATION_TIME"
: Separation time for knapsack cover.

"MSK_DINF_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.

"MSK_DINF_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 *"MSK_IINF_MIO_NUM_RELAX"* is strictly positive.

"MSK_DINF_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 *"MSK_IINF_MIO_NUM_INT_SOLUTIONS"*.

"MSK_DINF_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 *MSK_DPAR_MIO_REL_GAP_CONST*. Otherwise it has the value -1.0.

"MSK_DINF_MIO_PROBING_TIME"
: Total time for probing.

"MSK_DINF_MIO_ROOT_CUTGEN_TIME"
: Total time for cut generation.

"MSK_DINF_MIO_ROOT_OPTIMIZER_TIME"
: Time spent in the optimizer while solving the root node relaxation

"MSK_DINF_MIO_ROOT_PRESOLVE_TIME"
: Time spent presolving the problem at the root node.

"MSK_DINF_MIO_TIME"
: Time spent in the mixed-integer optimizer.

"MSK_DINF_MIO_USER_OBJ_CUT"
: If the objective cut is used, then this information item has the value of the cut.

"MSK_DINF_OPTIMIZER_TIME"
: Total time spent in the optimizer since it was invoked.

"MSK_DINF_PRESOLVE_ELI_TIME"
: Total time spent in the eliminator since the presolve was invoked.

"MSK_DINF_PRESOLVE_LINDEP_TIME"
: Total time spent in the linear dependency checker since the presolve was invoked.

"MSK_DINF_PRESOLVE_TIME"
: Total time (in seconds) spent in the presolve since it was invoked.

"MSK_DINF_PRIMAL_REPAIR_PENALTY_OBJ"
: The optimal objective value of the penalty function.

"MSK_DINF_QCQO_REFORMULATE_MAX_PERTURBATION"
: Maximum absolute diagonal perturbation occurring during the QCQO reformulation.

"MSK_DINF_QCQO_REFORMULATE_TIME"
: Time spent with conic quadratic reformulation.

"MSK_DINF_QCQO_REFORMULATE_WORST_CHOLESKY_COLUMN_SCALING"
: Worst Cholesky column scaling.

"MSK_DINF_QCQO_REFORMULATE_WORST_CHOLESKY_DIAG_SCALING"
: Worst Cholesky diagonal scaling.

"MSK_DINF_RD_TIME"
: Time spent reading the data file.

"MSK_DINF_SIM_DUAL_TIME"
: Time spent in the dual simplex optimizer since invoking it.

"MSK_DINF_SIM_FEAS"
: Feasibility measure reported by the simplex optimizer.

"MSK_DINF_SIM_OBJ"
: Objective value reported by the simplex optimizer.

"MSK_DINF_SIM_PRIMAL_TIME"
: Time spent in the primal simplex optimizer since invoking it.

"MSK_DINF_SIM_TIME"
: Time spent in the simplex optimizer since invoking it.

"MSK_DINF_SOL_BAS_DUAL_OBJ"
: Dual objective value of the basic solution. Updated if *MSK_IPAR_AUTO_UPDATE_SOL_INFO* is set .

"MSK_DINF_SOL_BAS_DVIOLCON"
: Maximal dual bound violation for x^c in the basic solution. Updated if *MSK_IPAR_AUTO_UPDATE_SOL_INFO* is set .

"MSK_DINF_SOL_BAS_DVIOLVAR"
: Maximal dual bound violation for x^x in the basic solution. Updated if *MSK_IPAR_AUTO_UPDATE_SOL_INFO* is set .

"MSK_DINF_SOL_BAS_NRM_BARX"
Infinity norm of $\overline{X}$ in the basic solution.

"MSK_DINF_SOL_BAS_NRM_SLC"
Infinity norm of s_l^c in the basic solution.

"MSK_DINF_SOL_BAS_NRM_SLX"
Infinity norm of s_l^x in the basic solution.

"MSK_DINF_SOL_BAS_NRM_SUC"
Infinity norm of s_u^c in the basic solution.

"MSK_DINF_SOL_BAS_NRM_SUX"
Infinity norm of s_u^X in the basic solution.

"MSK_DINF_SOL_BAS_NRM_XC"
Infinity norm of x^c in the basic solution.

"MSK_DINF_SOL_BAS_NRM_XX"
Infinity norm of x^x in the basic solution.

"MSK_DINF_SOL_BAS_NRM_Y"
Infinity norm of y in the basic solution.

"MSK_DINF_SOL_BAS_PRIMAL_OBJ"
Primal objective value of the basic solution. Updated if *MSK_IPAR_AUTO_UPDATE_SOL_INFO* is set .

"MSK_DINF_SOL_BAS_PVIOLCON"
Maximal primal bound violation for x^c in the basic solution. Updated if *MSK_IPAR_AUTO_UPDATE_SOL_INFO* is set .

"MSK_DINF_SOL_BAS_PVIOLVAR"
Maximal primal bound violation for x^x in the basic solution. Updated if *MSK_IPAR_AUTO_UPDATE_SOL_INFO* is set .

"MSK_DINF_SOL_ITG_NRM_BARX"
Infinity norm of $\overline{X}$ in the integer solution.

"MSK_DINF_SOL_ITG_NRM_XC"
Infinity norm of x^c in the integer solution.

"MSK_DINF_SOL_ITG_NRM_XX"
Infinity norm of x^x in the integer solution.

"MSK_DINF_SOL_ITG_PRIMAL_OBJ"
Primal objective value of the integer solution. Updated if *MSK_IPAR_AUTO_UPDATE_SOL_INFO* is set .

"MSK_DINF_SOL_ITG_PVIOLBARVAR"
Maximal primal bound violation for $\overline{X}$ in the integer solution. Updated if *MSK_IPAR_AUTO_UPDATE_SOL_INFO* is set .

"MSK_DINF_SOL_ITG_PVIOLCON"
Maximal primal bound violation for x^c in the integer solution. Updated if *MSK_IPAR_AUTO_UPDATE_SOL_INFO* is set .

"MSK_DINF_SOL_ITG_PVIOLCONES"
Maximal primal violation for primal conic constraints in the integer solution. Updated if *MSK_IPAR_AUTO_UPDATE_SOL_INFO* is set .

"MSK_DINF_SOL_ITG_PVIOLITG"
Maximal violation for the integer constraints in the integer solution. Updated if *MSK_IPAR_AUTO_UPDATE_SOL_INFO* is set .

"MSK_DINF_SOL_ITG_PVIOLVAR"
Maximal primal bound violation for x^x in the integer solution. Updated if *MSK_IPAR_AUTO_UPDATE_SOL_INFO* is set .

"MSK_DINF_SOL_ITR_DUAL_OBJ"
Dual objective value of the interior-point solution. Updated if *MSK_IPAR_AUTO_UPDATE_SOL_INFO* is set .

"MSK_DINF_SOL_ITR_DVIOLBARVAR"
: Maximal dual bound violation for $\overline{X}$ in the interior-point solution. Updated if *MSK_IPAR_AUTO_UPDATE_SOL_INFO* is set .

"MSK_DINF_SOL_ITR_DVIOLCON"
: Maximal dual bound violation for x^c in the interior-point solution. Updated if *MSK_IPAR_AUTO_UPDATE_SOL_INFO* is set .

"MSK_DINF_SOL_ITR_DVIOLCONES"
: Maximal dual violation for dual conic constraints in the interior-point solution. Updated if *MSK_IPAR_AUTO_UPDATE_SOL_INFO* is set .

"MSK_DINF_SOL_ITR_DVIOLVAR"
: Maximal dual bound violation for x^x in the interior-point solution. Updated if *MSK_IPAR_AUTO_UPDATE_SOL_INFO* is set .

"MSK_DINF_SOL_ITR_NRM_BARS"
: Infinity norm of $\overline{S}$ in the interior-point solution.

"MSK_DINF_SOL_ITR_NRM_BARX"
: Infinity norm of $\overline{X}$ in the interior-point solution.

"MSK_DINF_SOL_ITR_NRM_SLC"
: Infinity norm of s_l^c in the interior-point solution.

"MSK_DINF_SOL_ITR_NRM_SLX"
: Infinity norm of s_l^x in the interior-point solution.

"MSK_DINF_SOL_ITR_NRM_SNX"
: Infinity norm of s_n^x in the interior-point solution.

"MSK_DINF_SOL_ITR_NRM_SUC"
: Infinity norm of s_u^c in the interior-point solution.

"MSK_DINF_SOL_ITR_NRM_SUX"
: Infinity norm of s_u^X in the interior-point solution.

"MSK_DINF_SOL_ITR_NRM_XC"
: Infinity norm of x^c in the interior-point solution.

"MSK_DINF_SOL_ITR_NRM_XX"
: Infinity norm of x^x in the interior-point solution.

"MSK_DINF_SOL_ITR_NRM_Y"
: Infinity norm of y in the interior-point solution.

"MSK_DINF_SOL_ITR_PRIMAL_OBJ"
: Primal objective value of the interior-point solution. Updated if *MSK_IPAR_AUTO_UPDATE_SOL_INFO* is set .

"MSK_DINF_SOL_ITR_PVIOLBARVAR"
: Maximal primal bound violation for $\overline{X}$ in the interior-point solution. Updated if *MSK_IPAR_AUTO_UPDATE_SOL_INFO* is set .

"MSK_DINF_SOL_ITR_PVIOLCON"
: Maximal primal bound violation for x^c in the interior-point solution. Updated if *MSK_IPAR_AUTO_UPDATE_SOL_INFO* is set .

"MSK_DINF_SOL_ITR_PVIOLCONES"
: Maximal primal violation for primal conic constraints in the interior-point solution. Updated if *MSK_IPAR_AUTO_UPDATE_SOL_INFO* is set .

"MSK_DINF_SOL_ITR_PVIOLVAR"
: Maximal primal bound violation for x^x in the interior-point solution. Updated if *MSK_IPAR_AUTO_UPDATE_SOL_INFO* is set .

"MSK_DINF_TO_CONIC_TIME"
: Time spent in the last to conic reformulation.

feature
: License feature

"MSK_FEATURE_PTS"
Base system.

"MSK_FEATURE_PTON"
Conic extension.

liinfitem

Long integer information items.

"MSK_LIINF_BI_CLEAN_DUAL_DEG_ITER"
Number of dual degenerate clean iterations performed in the basis identification.

"MSK_LIINF_BI_CLEAN_DUAL_ITER"
Number of dual clean iterations performed in the basis identification.

"MSK_LIINF_BI_CLEAN_PRIMAL_DEG_ITER"
Number of primal degenerate clean iterations performed in the basis identification.

"MSK_LIINF_BI_CLEAN_PRIMAL_ITER"
Number of primal clean iterations performed in the basis identification.

"MSK_LIINF_BI_DUAL_ITER"
Number of dual pivots performed in the basis identification.

"MSK_LIINF_BI_PRIMAL_ITER"
Number of primal pivots performed in the basis identification.

"MSK_LIINF_INTPNT_FACTOR_NUM_NZ"
Number of non-zeros in factorization.

"MSK_LIINF_MIO_ANZ"
Number of non-zero entries in the constraint matrix of the probelm to be solved by the mixed-integer optimizer.

"MSK_LIINF_MIO_INTPNT_ITER"
Number of interior-point iterations performed by the mixed-integer optimizer.

"MSK_LIINF_MIO_PRESOLVED_ANZ"
Number of non-zero entries in the constraint matrix of the problem after the mixed-integer optimizer's presolve.

"MSK_LIINF_MIO_SIMPLEX_ITER"
Number of simplex iterations performed by the mixed-integer optimizer.

"MSK_LIINF_RD_NUMANZ"
Number of non-zeros in A that is read.

"MSK_LIINF_RD_NUMQNZ"
Number of Q non-zeros.

iinfitem

Integer information items.

"MSK_IINF_ANA_PRO_NUM_CON"
Number of constraints in the problem.

"MSK_IINF_ANA_PRO_NUM_CON_EQ"
Number of equality constraints.

"MSK_IINF_ANA_PRO_NUM_CON_FR"
Number of unbounded constraints.

"MSK_IINF_ANA_PRO_NUM_CON_LO"
Number of constraints with a lower bound and an infinite upper bound.

"MSK_IINF_ANA_PRO_NUM_CON_RA"
Number of constraints with finite lower and upper bounds.

"MSK_IINF_ANA_PRO_NUM_CON_UP"
Number of constraints with an upper bound and an infinite lower bound.

"MSK_IINF_ANA_PRO_NUM_VAR"
Number of variables in the problem.

"MSK_IINF_ANA_PRO_NUM_VAR_BIN"
Number of binary (0-1) variables.

"MSK_IINF_ANA_PRO_NUM_VAR_CONT"
Number of continuous variables.

"MSK_IINF_ANA_PRO_NUM_VAR_EQ"
Number of fixed variables.

"MSK_IINF_ANA_PRO_NUM_VAR_FR"
Number of free variables.

"MSK_IINF_ANA_PRO_NUM_VAR_INT"
Number of general integer variables.

"MSK_IINF_ANA_PRO_NUM_VAR_LO"
Number of variables with a lower bound and an infinite upper bound.

"MSK_IINF_ANA_PRO_NUM_VAR_RA"
Number of variables with finite lower and upper bounds.

"MSK_IINF_ANA_PRO_NUM_VAR_UP"
Number of variables with an upper bound and an infinite lower bound.

"MSK_IINF_INTPNT_FACTOR_DIM_DENSE"
Dimension of the dense sub system in factorization.

"MSK_IINF_INTPNT_ITER"
Number of interior-point iterations since invoking the interior-point optimizer.

"MSK_IINF_INTPNT_NUM_THREADS"
Number of threads that the interior-point optimizer is using.

"MSK_IINF_INTPNT_SOLVE_DUAL"
Non-zero if the interior-point optimizer is solving the dual problem.

"MSK_IINF_MIO_ABSGAP_SATISFIED"
Non-zero if absolute gap is within tolerances.

"MSK_IINF_MIO_CLIQUE_TABLE_SIZE"
Size of the clique table.

"MSK_IINF_MIO_CONSTRUCT_SOLUTION"
This item informs if **MOSEK** constructed an initial integer feasible solution.

- -1: tried, but failed,
- 0: no partial solution supplied by the user,
- 1: constructed feasible solution.

"MSK_IINF_MIO_NODE_DEPTH"
Depth of the last node solved.

"MSK_IINF_MIO_NUM_ACTIVE_NODES"
Number of active branch and bound nodes.

"MSK_IINF_MIO_NUM_BRANCH"
Number of branches performed during the optimization.

"MSK_IINF_MIO_NUM_CLIQUE_CUTS"
Number of clique cuts.

"MSK_IINF_MIO_NUM_CMIR_CUTS"
Number of Complemented Mixed Integer Rounding (CMIR) cuts.

"MSK_IINF_MIO_NUM_GOMORY_CUTS"
Number of Gomory cuts.

"MSK_IINF_MIO_NUM_IMPLIED_BOUND_CUTS"
Number of implied bound cuts.

"MSK_IINF_MIO_NUM_INT_SOLUTIONS"
Number of integer feasible solutions that have been found.

"MSK_IINF_MIO_NUM_KNAPSACK_COVER_CUTS"
Number of clique cuts.

"MSK_IINF_MIO_NUM_RELAX"
: Number of relaxations solved during the optimization.

"MSK_IINF_MIO_NUM_REPEATED_PRESOLVE"
: Number of times presolve was repeated at root.

"MSK_IINF_MIO_NUMBIN"
: Number of binary variables in the problem to be solved by the mixed-integer optimizer.

"MSK_IINF_MIO_NUMBINCONEVAR"
: Number of binary cone variables in the problem to be solved by the mixed-integer optimizer.

"MSK_IINF_MIO_NUMCON"
: Number of constraints in the problem to be solved by the mixed-integer optimizer.

"MSK_IINF_MIO_NUMCONE"
: Number of cones in the problem to be solved by the mixed-integer optimizer.

"MSK_IINF_MIO_NUMCONEVAR"
: Number of cone variables in the problem to be solved by the mixed-integer optimizer.

"MSK_IINF_MIO_NUMCONT"
: Number of continuous variables in the problem to be solved by the mixed-integer optimizer.

"MSK_IINF_MIO_NUMCONTCONEVAR"
: Number of continuous cone variables in the problem to be solved by the mixed-integer optimizer.

"MSK_IINF_MIO_NUMDEXPCONES"
: Number of dual exponential cones in the problem to be solved by the mixed-integer optimizer.

"MSK_IINF_MIO_NUMDPOWCONES"
: Number of dual power cones in the problem to be solved by the mixed-integer optimizer.

"MSK_IINF_MIO_NUMINT"
: Number of integer variables in the problem to be solved by the mixed-integer optimizer.

"MSK_IINF_MIO_NUMINTCONEVAR"
: Number of integer cone variables in the problem to be solved by the mixed-integer optimizer.

"MSK_IINF_MIO_NUMPEXPCONES"
: Number of primal exponential cones in the problem to be solved by the mixed-integer optimizer.

"MSK_IINF_MIO_NUMPPOWCONES"
: Number of primal power cones in the problem to be solved by the mixed-integer optimizer.

"MSK_IINF_MIO_NUMQCONES"
: Number of quadratic cones in the problem to be solved by the mixed-integer optimizer.

"MSK_IINF_MIO_NUMRQCONES"
: Number of rotated quadratic cones in the problem to be solved by the mixed-integer optimizer.

"MSK_IINF_MIO_NUMVAR"
: Number of variables in the problem to be solved by the mixed-integer optimizer.

"MSK_IINF_MIO_OBJ_BOUND_DEFINED"
: Non-zero if a valid objective bound has been found, otherwise zero.

"MSK_IINF_MIO_PRESOLVED_NUMBIN"
: Number of binary variables in the problem after the mixed-integer optimizer's presolve.

"MSK_IINF_MIO_PRESOLVED_NUMBINCONEVAR"
: Number of binary cone variables in the problem after the mixed-integer optimizer's presolve.

"MSK_IINF_MIO_PRESOLVED_NUMCON"
: Number of constraints in the problem after the mixed-integer optimizer's presolve.

"MSK_IINF_MIO_PRESOLVED_NUMCONE"
: Number of cones in the problem after the mixed-integer optimizer's presolve.

"MSK_IINF_MIO_PRESOLVED_NUMCONEVAR"
: Number of cone variables in the problem after the mixed-integer optimizer's presolve.

"MSK_IINF_MIO_PRESOLVED_NUMCONT"
: Number of continuous variables in the problem after the mixed-integer optimizer's presolve.

"MSK_IINF_MIO_PRESOLVED_NUMCONTCONEVAR"
: Number of continuous cone variables in the problem after the mixed-integer optimizer's presolve.

"MSK_IINF_MIO_PRESOLVED_NUMDEXPCONES"
: Number of dual exponential cones in the problem after the mixed-integer optimizer's presolve.

"MSK_IINF_MIO_PRESOLVED_NUMDPOWCONES"
: Number of dual power cones in the problem after the mixed-integer optimizer's presolve.

"MSK_IINF_MIO_PRESOLVED_NUMINT"
: Number of integer variables in the problem after the mixed-integer optimizer's presolve.

"MSK_IINF_MIO_PRESOLVED_NUMINTCONEVAR"
: Number of integer cone variables in the problem after the mixed-integer optimizer's presolve.

"MSK_IINF_MIO_PRESOLVED_NUMPEXPCONES"
: Number of primal exponential cones in the problem after the mixed-integer optimizer's presolve.

"MSK_IINF_MIO_PRESOLVED_NUMPPOWCONES"
: Number of primal power cones in the problem after the mixed-integer optimizer's presolve.

"MSK_IINF_MIO_PRESOLVED_NUMQCONES"
: Number of quadratic cones in the problem after the mixed-integer optimizer's presolve.

"MSK_IINF_MIO_PRESOLVED_NUMRQCONES"
: Number of rotated quadratic cones in the problem after the mixed-integer optimizer's presolve.

"MSK_IINF_MIO_PRESOLVED_NUMVAR"
: Number of variables in the problem after the mixed-integer optimizer's presolve.

"MSK_IINF_MIO_RELGAP_SATISFIED"
: Non-zero if relative gap is within tolerances.

"MSK_IINF_MIO_TOTAL_NUM_CUTS"
: Total number of cuts generated by the mixed-integer optimizer.

"MSK_IINF_MIO_USER_OBJ_CUT"
: If it is non-zero, then the objective cut is used.

"MSK_IINF_OPT_NUMCON"
: Number of constraints in the problem solved when the optimizer is called.

"MSK_IINF_OPT_NUMVAR"
: Number of variables in the problem solved when the optimizer is called

"MSK_IINF_OPTIMIZE_RESPONSE"
: The response code returned by optimize.

"MSK_IINF_PURIFY_DUAL_SUCCESS"
: Is nonzero if the dual solution is purified.

"MSK_IINF_PURIFY_PRIMAL_SUCCESS"
: Is nonzero if the primal solution is purified.

"MSK_IINF_RD_NUMBARVAR"
: Number of symmetric variables read.

"MSK_IINF_RD_NUMCON"
: Number of constraints read.

"MSK_IINF_RD_NUMCONE"
: Number of conic constraints read.

"MSK_IINF_RD_NUMINTVAR"
: Number of integer-constrained variables read.

"MSK_IINF_RD_NUMQ"
: Number of nonempty Q matrices read.

"MSK_IINF_RD_NUMVAR"
: Number of variables read.

"MSK_IINF_RD_PROTYPE"
: Problem type.

"MSK_IINF_SIM_DUAL_DEG_ITER"
: The number of dual degenerate iterations.

"MSK_IINF_SIM_DUAL_HOTSTART"
: If 1 then the dual simplex algorithm is solving from an advanced basis.

"MSK_IINF_SIM_DUAL_HOTSTART_LU"
: If 1 then a valid basis factorization of full rank was located and used by the dual simplex algorithm.

"MSK_IINF_SIM_DUAL_INF_ITER"
: The number of iterations taken with dual infeasibility.

"MSK_IINF_SIM_DUAL_ITER"
: Number of dual simplex iterations during the last optimization.

"MSK_IINF_SIM_NUMCON"
: Number of constraints in the problem solved by the simplex optimizer.

"MSK_IINF_SIM_NUMVAR"
: Number of variables in the problem solved by the simplex optimizer.

"MSK_IINF_SIM_PRIMAL_DEG_ITER"
: The number of primal degenerate iterations.

"MSK_IINF_SIM_PRIMAL_HOTSTART"
: If 1 then the primal simplex algorithm is solving from an advanced basis.

"MSK_IINF_SIM_PRIMAL_HOTSTART_LU"
: If 1 then a valid basis factorization of full rank was located and used by the primal simplex algorithm.

"MSK_IINF_SIM_PRIMAL_INF_ITER"
: The number of iterations taken with primal infeasibility.

"MSK_IINF_SIM_PRIMAL_ITER"
: Number of primal simplex iterations during the last optimization.

"MSK_IINF_SIM_SOLVE_DUAL"
: Is non-zero if dual problem is solved.

"MSK_IINF_SOL_BAS_PROSTA"
: Problem status of the basic solution. Updated after each optimization.

"MSK_IINF_SOL_BAS_SOLSTA"
: Solution status of the basic solution. Updated after each optimization.

"MSK_IINF_SOL_ITG_PROSTA"
: Problem status of the integer solution. Updated after each optimization.

"MSK_IINF_SOL_ITG_SOLSTA"
: Solution status of the integer solution. Updated after each optimization.

"MSK_IINF_SOL_ITR_PROSTA"
: Problem status of the interior-point solution. Updated after each optimization.

"MSK_IINF_SOL_ITR_SOLSTA"
: Solution status of the interior-point solution. Updated after each optimization.

"MSK_IINF_STO_NUM_A_REALLOC"
: Number of times the storage for storing A has been changed. A large value may indicates that memory fragmentation may occur.

inftype
: Information item types

"MSK_INF_DOU_TYPE"
: Is a double information type.

"MSK_INF_INT_TYPE"
: Is an integer.

"MSK_INF_LINT_TYPE"
: Is a long integer.

iomode
: Input/output modes

"MSK_IOMODE_READ"
: The file is read-only.

"MSK_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.

"MSK_IOMODE_READWRITE"
: The file is to read and write.

branchdir
: Specifies the branching direction.

"MSK_BRANCH_DIR_FREE"
: The mixed-integer optimizer decides which branch to choose.

"MSK_BRANCH_DIR_UP"
: The mixed-integer optimizer always chooses the up branch first.

"MSK_BRANCH_DIR_DOWN"
: The mixed-integer optimizer always chooses the down branch first.

"MSK_BRANCH_DIR_NEAR"
: Branch in direction nearest to selected fractional variable.

"MSK_BRANCH_DIR_FAR"
: Branch in direction farthest from selected fractional variable.

"MSK_BRANCH_DIR_ROOT_LP"
: Chose direction based on root lp value of selected variable.

"MSK_BRANCH_DIR_GUIDED"
: Branch in direction of current incumbent.

"MSK_BRANCH_DIR_PSEUDOCOST"
: Branch based on the pseudocost of the variable.

miocontsoltype
: Continuous mixed-integer solution type

"MSK_MIO_CONT_SOL_NONE"
: No interior-point or basic solution are reported when the mixed-integer optimizer is used.

"MSK_MIO_CONT_SOL_ROOT"
: The reported interior-point and basic solutions are a solution to the root node problem when mixed-integer optimizer is used.

"MSK_MIO_CONT_SOL_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.

"MSK_MIO_CONT_SOL_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

"MSK_MIO_MODE_IGNORED"
: The integer constraints are ignored and the problem is solved as a continuous problem.

"MSK_MIO_MODE_SATISFIED"
: Integer restrictions should be satisfied.

mionodeseltype
Mixed-integer node selection types

"MSK_MIO_NODE_SELECTION_FREE"
The optimizer decides the node selection strategy.

"MSK_MIO_NODE_SELECTION_FIRST"
The optimizer employs a depth first node selection strategy.

"MSK_MIO_NODE_SELECTION_BEST"
The optimizer employs a best bound node selection strategy.

"MSK_MIO_NODE_SELECTION_PSEUDO"
The optimizer employs selects the node based on a pseudo cost estimate.

mpsformat
MPS file format type

"MSK_MPS_FORMAT_STRICT"
It is assumed that the input file satisfies the MPS format strictly.

"MSK_MPS_FORMAT_RELAXED"
It is assumed that the input file satisfies a slightly relaxed version of the MPS format.

"MSK_MPS_FORMAT_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.

"MSK_MPS_FORMAT_CPLEX"
The CPLEX compatible version of the MPS format is employed.

objsense
Objective sense types

"MSK_OBJECTIVE_SENSE_MINIMIZE"
The problem should be minimized.

"MSK_OBJECTIVE_SENSE_MAXIMIZE"
The problem should be maximized.

onoffkey
On/off

"MSK_ON"
Switch the option on.

"MSK_OFF"
Switch the option off.

optimizertype
Optimizer types

"MSK_OPTIMIZER_CONIC"
The optimizer for problems having conic constraints.

"MSK_OPTIMIZER_DUAL_SIMPLEX"
The dual simplex optimizer is used.

"MSK_OPTIMIZER_FREE"
The optimizer is chosen automatically.

"MSK_OPTIMIZER_FREE_SIMPLEX"
One of the simplex optimizers is used.

"MSK_OPTIMIZER_INTPNT"
The interior-point optimizer is used.

"MSK_OPTIMIZER_MIXED_INT"
The mixed-integer optimizer.

"MSK_OPTIMIZER_PRIMAL_SIMPLEX"
The primal simplex optimizer is used.

orderingtype
Ordering strategies

"MSK_ORDER_METHOD_FREE"
: The ordering method is chosen automatically.

"MSK_ORDER_METHOD_APPMINLOC"
: Approximate minimum local fill-in ordering is employed.

"MSK_ORDER_METHOD_EXPERIMENTAL"
: This option should not be used.

"MSK_ORDER_METHOD_TRY_GRAPHPAR"
: Always try the graph partitioning based ordering.

"MSK_ORDER_METHOD_FORCE_GRAPHPAR"
: Always use the graph partitioning based ordering even if it is worse than the approximate minimum local fill ordering.

"MSK_ORDER_METHOD_NONE"
: No ordering is used.

`presolvemode`
: Presolve method.

"MSK_PRESOLVE_MODE_OFF"
: The problem is not presolved before it is optimized.

"MSK_PRESOLVE_MODE_ON"
: The problem is presolved before it is optimized.

"MSK_PRESOLVE_MODE_FREE"
: It is decided automatically whether to presolve before the problem is optimized.

`parametertype`
: Parameter type

"MSK_PAR_INVALID_TYPE"
: Not a valid parameter.

"MSK_PAR_DOU_TYPE"
: Is a double parameter.

"MSK_PAR_INT_TYPE"
: Is an integer parameter.

"MSK_PAR_STR_TYPE"
: Is a string parameter.

`problemitem`
: Problem data items

"MSK_PI_VAR"
: Item is a variable.

"MSK_PI_CON"
: Item is a constraint.

"MSK_PI_CONE"
: Item is a cone.

`problemtype`
: Problem types

"MSK_PROBTYPE_LO"
: The problem is a linear optimization problem.

"MSK_PROBTYPE_QO"
: The problem is a quadratic optimization problem.

"MSK_PROBTYPE_QCQO"
: The problem is a quadratically constrained optimization problem.

"MSK_PROBTYPE_CONIC"
: A conic optimization.

"MSK_PROBTYPE_MIXED"
: General nonlinear constraints and conic constraints. This combination can not be solved by **MOSEK**.

prosta
Problem status keys

"MSK_PRO_STA_UNKNOWN"
Unknown problem status.

"MSK_PRO_STA_PRIM_AND_DUAL_FEAS"
The problem is primal and dual feasible.

"MSK_PRO_STA_PRIM_FEAS"
The problem is primal feasible.

"MSK_PRO_STA_DUAL_FEAS"
The problem is dual feasible.

"MSK_PRO_STA_PRIM_INFEAS"
The problem is primal infeasible.

"MSK_PRO_STA_DUAL_INFEAS"
The problem is dual infeasible.

"MSK_PRO_STA_PRIM_AND_DUAL_INFEAS"
The problem is primal and dual infeasible.

"MSK_PRO_STA_ILL_POSED"
The problem is ill-posed. For example, it may be primal and dual feasible but have a positive duality gap.

"MSK_PRO_STA_PRIM_INFEAS_OR_UNBOUNDED"
The problem is either primal infeasible or unbounded. This may occur for mixed-integer problems.

xmlwriteroutputtype
XML writer output mode

"MSK_WRITE_XML_MODE_ROW"
Write in row order.

"MSK_WRITE_XML_MODE_COL"
Write in column order.

rescodetype
Response code type

"MSK_RESPONSE_OK"
The response code is OK.

"MSK_RESPONSE_WRN"
The response code is a warning.

"MSK_RESPONSE_TRM"
The response code is an optimizer termination status.

"MSK_RESPONSE_ERR"
The response code is an error.

"MSK_RESPONSE_UNK"
The response code does not belong to any class.

scalingtype
Scaling type

"MSK_SCALING_FREE"
The optimizer chooses the scaling heuristic.

"MSK_SCALING_NONE"
No scaling is performed.

"MSK_SCALING_MODERATE"
A conservative scaling is performed.

"MSK_SCALING_AGGRESSIVE"
A very aggressive scaling is performed.

scalingmethod
Scaling method

"MSK_SCALING_METHOD_POW2"
: Scales only with power of 2 leaving the mantissa untouched.

"MSK_SCALING_METHOD_FREE"
: The optimizer chooses the scaling heuristic.

`sensitivitytype`
Sensitivity types

"MSK_SENSITIVITY_TYPE_BASIS"
: Basis sensitivity analysis is performed.

`simseltype`
Simplex selection strategy

"MSK_SIM_SELECTION_FREE"
: The optimizer chooses the pricing strategy.

"MSK_SIM_SELECTION_FULL"
: The optimizer uses full pricing.

"MSK_SIM_SELECTION_ASE"
: The optimizer uses approximate steepest-edge pricing.

"MSK_SIM_SELECTION_DEVEX"
: The optimizer uses devex steepest-edge pricing (or if it is not available an approximate steep-edge selection).

"MSK_SIM_SELECTION_SE"
: The optimizer uses steepest-edge selection (or if it is not available an approximate steep-edge selection).

"MSK_SIM_SELECTION_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

"MSK_SOL_ITEM_XC"
: Solution for the constraints.

"MSK_SOL_ITEM_XX"
: Variable solution.

"MSK_SOL_ITEM_Y"
: Lagrange multipliers for equations.

"MSK_SOL_ITEM_SLC"
: Lagrange multipliers for lower bounds on the constraints.

"MSK_SOL_ITEM_SUC"
: Lagrange multipliers for upper bounds on the constraints.

"MSK_SOL_ITEM_SLX"
: Lagrange multipliers for lower bounds on the variables.

"MSK_SOL_ITEM_SUX"
: Lagrange multipliers for upper bounds on the variables.

"MSK_SOL_ITEM_SNX"
: Lagrange multipliers corresponding to the conic constraints on the variables.

`solsta`
Solution status keys

"MSK_SOL_STA_UNKNOWN"
: Status of the solution is unknown.

"MSK_SOL_STA_OPTIMAL"
: The solution is optimal.

"MSK_SOL_STA_PRIM_FEAS"
: The solution is primal feasible.

"MSK_SOL_STA_DUAL_FEAS"
The solution is dual feasible.

"MSK_SOL_STA_PRIM_AND_DUAL_FEAS"
The solution is both primal and dual feasible.

"MSK_SOL_STA_PRIM_INFEAS_CER"
The solution is a certificate of primal infeasibility.

"MSK_SOL_STA_DUAL_INFEAS_CER"
The solution is a certificate of dual infeasibility.

"MSK_SOL_STA_PRIM_ILLPOSED_CER"
The solution is a certificate that the primal problem is illposed.

"MSK_SOL_STA_DUAL_ILLPOSED_CER"
The solution is a certificate that the dual problem is illposed.

"MSK_SOL_STA_INTEGER_OPTIMAL"
The primal solution is integer optimal.

soltype
Solution types

"MSK_SOL_BAS"
The basic solution.

"MSK_SOL_ITR"
The interior solution.

"MSK_SOL_ITG"
The integer solution.

solveform
Solve primal or dual form

"MSK_SOLVE_FREE"
The optimizer is free to solve either the primal or the dual problem.

"MSK_SOLVE_PRIMAL"
The optimizer should solve the primal problem.

"MSK_SOLVE_DUAL"
The optimizer should solve the dual problem.

stakey
Status keys

"MSK_SK_UNK"
The status for the constraint or variable is unknown.

"MSK_SK_BAS"
The constraint or variable is in the basis.

"MSK_SK_SUPBAS"
The constraint or variable is super basic.

"MSK_SK_LOW"
The constraint or variable is at its lower bound.

"MSK_SK_UPR"
The constraint or variable is at its upper bound.

"MSK_SK_FIX"
The constraint or variable is fixed.

"MSK_SK_INF"
The constraint or variable is infeasible in the bounds.

startpointtype
Starting point types

"MSK_STARTING_POINT_FREE"
The starting point is chosen automatically.

"MSK_STARTING_POINT_GUESS"
The optimizer guesses a starting point.

"MSK_STARTING_POINT_CONSTANT"
: The optimizer constructs a starting point by assigning a constant value to all primal and dual variables. This starting point is normally robust.

"MSK_STARTING_POINT_SATISFY_BOUNDS"
: The starting point is chosen to satisfy all the simple bounds on nonlinear variables. If this starting point is employed, then more care than usual should employed when choosing the bounds on the nonlinear variables. In particular very tight bounds should be avoided.

`streamtype`
Stream types

"MSK_STREAM_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.

"MSK_STREAM_MSG"
: Message stream. Log information relating to performance and progress of the optimization is written to this stream.

"MSK_STREAM_ERR"
: Error stream. Error messages are written to this stream.

"MSK_STREAM_WRN"
: Warning stream. Warning messages are written to this stream.

`value`
Integer values

"MSK_MAX_STR_LEN"
: Maximum string length allowed in **MOSEK**.

"MSK_LICENSE_BUFFER_LENGTH"
: The length of a license key buffer.

`variabletype`
Variable types

"MSK_VAR_TYPE_CONT"
: Is a continuous variable.

"MSK_VAR_TYPE_INT"
: Is an integer variable.

15.8 Nonlinear interfaces (obsolete)

Important: This is a legacy document for users familiar with SCopt, DGopt, EXPopt, mskenopt, mskscopt and mskgpopt from previous versions of **MOSEK**. These interfaces have now been removed. We assume familiarity with documentation included in version 8. All problems expressible with this interface can (and should) be reformulated using the exponential cone and power cones.

New users should formulate problems involving powers, logarithms and exponentials directly in conic form.

Conversion tutorial

We recommend converting all nonlinear problems using SCopt, DGopt, EXPopt, mskenopt, mskscopt and mskgpopt into conic form. Depending on the values of f, g, h either the epigraph or hypograph of a SCopt function if convex, and a bounding variable can be introduced following the basic rules below. We assume all variables are within safe bounds where the SCopt operators are defined and convex. We also assume $f > 0$.

A more comprehensive modeling guide for these types of problems can be found in the **MOSEK** Modeling Cookbook.

Powers

Consider $f(x+h)^g$. This can be reformulated using the power cone.

- If $g > 1$ then $t \geq f(x+h)^g$ is equivalent to $(t/f)^{1/g} \geq |x+h|$, that is $(t/f, 1, x+h) \in \mathcal{P}_3^{1/g, 1-1/g}$.
- If $0 < g < 1$ then $|t| \leq f(x+h)^g$ is equivalent to $(x+h, 1, t/f) \in \mathcal{P}_3^{g, 1-g}$.
- If $g < 0$ then $t \geq f(x+h)^g$ is equivalent to $(t/f)(x+h)^{-g} \geq 1$, that is $(t/f, x+h, 1) \in \mathcal{P}_3^{1/(1-g), -g/(1-g)}$.

Logarithm

The bound $t \leq f \log(gx+h)$ is equivalent to $(gx+h, 1, t/f) \in K_{\exp}$.

Entropy

The bound $t \geq fx \log x$ is equivalent to $(1, x, -t/f) \in K_{\exp}$.

Exponential

The bound $t \geq f \exp(gx+h)$ is equivalent to $(t/f, 1, gx+h) \in K_{\exp}$.

Exponential optimization (EXPopt), Geometric programming (mskgpopt)

For a basic tutorial in geometric programming (GP) see Sec. 6.8.
An exponential optimization problem in standard form consists of constraints of the type:

$$t \geq \log\left(\sum_i \exp(a_i^T x + b_i)\right).$$

This log-sum-exp bound is equivalent to

$$\sum_i \exp(a_i^T x + b_i - t) \leq 1$$

and requires bounding each exponential function as explained above.

Dual geometric optimization (DGopt)

The objective function of a dual geometric problem involves maximizing expressions of the form

$$x \log \frac{c}{x} \quad \text{and} \quad x_i \log \frac{e^T x}{x_i},$$

which can be achieved using bounds $t \leq x \log \frac{y}{x}$, that is $(t, x, y) \in K_{\exp}$.

Chapter 16

Supported File Formats

MOSEK supports a range of problem and solution formats listed in Table 16.1 and Table 16.2. The **Task format** is **MOSEK**'s native binary format and it supports all features that **MOSEK** supports. The **OPF format** is **MOSEK**'s human-readable alternative that supports nearly all features (everything except semidefinite problems). In general, text formats are significantly slower to read, but can be examined and edited directly in any text editor.

Problem formats

Table 16.1: List of supported file formats for optimization problems. The column *Conic* refers to conic problems involving the quadratic, rotated quadratic, power or exponential cone. The last two columns indicate if the format supports solutions and optimizer parameters.

Format Type	Ext.	Binary/Text	LP	QO	Conic	SDP	Sol	Param
LP	lp	plain text	X	X				
MPS	mps	plain text	X	X	X			
OPF	opf	plain text	X	X	X		X	X
PTF	ptf	plain text	X	X	X	X	X	
CBF	cbf	plain text	X		X	X		
Task format	task	binary	X	X	X	X	X	X
Jtask format	jtask	text	X	X	X	X	X	X

Solution formats

Table 16.2: List of supported solution formats.

Format Type	Ext.	Binary/Text	Description
SOL	sol	plain text	Interior Solution
	bas	plain text	Basic Solution
	int	plain text	Integer
Jsol format	jsol	text	Solution

Compression

MOSEK supports GZIP and Zstandard compression. Problem files with extension `.gz` (for GZIP) and `.zst` (for Zstandard) are assumed to be compressed when read, and are automatically compressed when written. For example, a file called

```
problem.mps.gz
```

will be considered as a GZIP compressed MPS file.

16.1 The LP File Format

MOSEK supports the LP file format with some extensions. The LP format is not a completely well-defined standard and hence different optimization packages may interpret the same LP file in slightly different ways. **MOSEK** tries to emulate as closely as possible CPLEX's behavior, but tries to stay backward compatible.

The LP file format can specify problems of the form

$$
\begin{array}{lccccc}
\text{minimize/maximize} & & & c^T x + \frac{1}{2} q^o(x) & & \\
\text{subject to} & l^c & \leq & Ax + \frac{1}{2} q(x) & \leq & u^c, \\
 & l^x & \leq & x & \leq & u^x, \\
 & & & x_{\mathcal{J}} \text{ integer}, & &
\end{array}
$$

where

- $x \in \mathbb{R}^n$ is the vector of decision variables.
- $c \in \mathbb{R}^n$ is the linear term in the objective.
- $q^o :\in \mathbb{R}^n \rightarrow \mathbb{R}$ is the quadratic term in the objective where

$$q^o(x) = x^T Q^o x$$

and it is assumed that

$$Q^o = (Q^o)^T.$$

- $A \in \mathbb{R}^{m \times n}$ is the constraint matrix.
- $l^c \in \mathbb{R}^m$ is the lower limit on the activity for the constraints.
- $u^c \in \mathbb{R}^m$ is the upper limit on the activity for the constraints.
- $l^x \in \mathbb{R}^n$ is the lower limit on the activity for the variables.
- $u^x \in \mathbb{R}^n$ is the upper limit on the activity for the variables.
- $q : \mathbb{R}^n \rightarrow \mathbb{R}$ is a vector of quadratic functions. Hence,

$$q_i(x) = x^T Q^i x$$

where it is assumed that

$$Q^i = (Q^i)^T.$$

- $\mathcal{J} \subseteq \{1, 2, \ldots, n\}$ is an index set of the integer constrained variables.

16.1.1 File Sections

An LP formatted file contains a number of sections specifying the objective, constraints, variable bounds, and variable types. The section keywords may be any mix of upper and lower case letters.

Objective Function

The first section beginning with one of the keywords

```
max
maximum
maximize
min
minimum
minimize
```

defines the objective sense and the objective function, i.e.

$$c^T x + \frac{1}{2} x^T Q^o x.$$

The objective may be given a name by writing

```
myname:
```

before the expressions. If no name is given, then the objective is named `obj`.

The objective function contains linear and quadratic terms. The linear terms are written as

```
4 x1 + x2 - 0.1 x3
```

and so forth. The quadratic terms are written in square brackets (`[ ]/2`) and are either squared or multiplied as in the examples

```
x1^2
```

and

```
x1 * x2
```

There may be zero or more pairs of brackets containing quadratic expressions.

An example of an objective section is

```
minimize
myobj: 4 x1 + x2 - 0.1 x3 + [ x1^2 + 2.1 x1 * x2 ]/2
```

Please note that the quadratic expressions are multiplied with $\frac{1}{2}$, so that the above expression means

$$\text{minimize} \quad 4x_1 + x_2 - 0.1 \cdot x_3 + \tfrac{1}{2}(x_1^2 + 2.1 \cdot x_1 \cdot x_2)$$

If the same variable occurs more than once in the linear part, the coefficients are added, so that `4 x1 + 2 x1` is equivalent to `6 x1`. In the quadratic expressions `x1 * x2` is equivalent to `x2 * x1` and, as in the linear part, if the same variables multiplied or squared occur several times their coefficients are added.

Constraints

The second section beginning with one of the keywords

```
subj to
subject to
s.t.
st
```

defines the linear constraint matrix A and the quadratic matrices Q^i.

A constraint contains a name (optional), expressions adhering to the same rules as in the objective and a bound:

```
subject to
con1: x1 + x2 + [ x3^2 ]/2 <= 5.1
```

The bound type (here `<=`) may be any of `<`, `<=`, `=`, `>`, `>=` (`<` and `<=` mean the same), and the bound may be any number.

In the standard LP format it is not possible to define more than one bound per line, but **MOSEK** supports defining ranged constraints by using double-colon (`::`) instead of a single-colon (`:`) after the constraint name, i.e.

$$-5 \leq x_1 + x_2 \leq 5 \tag{16.1}$$

may be written as

```
con:: -5 < x_1 + x_2 < 5
```

By default **MOSEK** writes ranged constraints this way.

If the files must adhere to the LP standard, ranged constraints must either be split into upper bounded and lower bounded constraints or be written as an equality with a slack variable. For example the expression (16.1) may be written as

$$x_1 + x_2 - sl_1 = 0,\ -5 \leq sl_1 \leq 5.$$

Bounds

Bounds on the variables can be specified in the bound section beginning with one of the keywords

```
bound
bounds
```

The bounds section is optional but should, if present, follow the `subject to` section. All variables listed in the bounds section must occur in either the objective or a constraint.

The default lower and upper bounds are 0 and $+\infty$. A variable may be declared free with the keyword `free`, which means that the lower bound is $-\infty$ and the upper bound is $+\infty$. Furthermore it may be assigned a finite lower and upper bound. The bound definitions for a given variable may be written in one or two lines, and bounds can be any number or $\pm\infty$ (written as `+inf`/`-inf`/`+infinity`/`-infinity`) as in the example

```
bounds
x1 free
x2 <= 5
0.1 <= x2
x3 = 42
2 <= x4 < +inf
```

Variable Types

The final two sections are optional and must begin with one of the keywords

```
bin
binaries
binary
```

and

```
gen
general
```

Under `general` all integer variables are listed, and under `binary` all binary (integer variables with bounds 0 and 1) are listed:

```
general
x1 x2
binary
x3 x4
```

Again, all variables listed in the binary or general sections must occur in either the objective or a constraint.

Terminating Section

Finally, an LP formatted file must be terminated with the keyword

```
end
```

16.1.2 LP File Examples

Linear example lo1.lp

```
\ File: lo1.lp
maximize
obj: 3 x1 + x2 + 5 x3 + x4
subject to
c1:  3 x1 + x2 + 2 x3 = 30
c2:  2 x1 + x2 + 3 x3 + x4 >= 15
c3:  2 x2 + 3 x4 <= 25
bounds
 0 <= x1 <= +infinity
 0 <= x2 <= 10
 0 <= x3 <= +infinity
 0 <= x4 <= +infinity
end
```

Mixed integer example milo1.lp

```
maximize
obj: x1 + 6.4e-01 x2
subject to
c1:  5e+01 x1 + 3.1e+01 x2 <= 2.5e+02
c2:  3e+00 x1 - 2e+00 x2 >= -4e+00
bounds
 0 <= x1 <= +infinity
 0 <= x2 <= +infinity
general
 x1 x2
end
```

16.1.3 LP Format peculiarities

Comments

Anything on a line after a \ is ignored and is treated as a comment.

Names

A name for an objective, a constraint or a variable may contain the letters `a-z`, `A-Z`, the digits `0-9` and the characters

```
!"#$%&()/,.;?@_'`|~
```

The first character in a name must not be a number, a period or the letter `e` or `E`. Keywords must not be used as names.

MOSEK accepts any character as valid for names, except `\0`. A name that is not allowed in LP file will be changed and a warning will be issued.

The algorithm for making names LP valid works as follows: The name is interpreted as an `utf-8` string. For a Unicode character `c`:

- If `c==_` (underscore), the output is `__` (two underscores).
- If `c` is a valid LP name character, the output is just `c`.
- If `c` is another character in the ASCII range, the output is `_XX`, where `XX` is the hexadecimal code for the character.
- If `c` is a character in the range *127-65535*, the output is `_uXXXX`, where `XXXX` is the hexadecimal code for the character.

- If `c` is a character above 65535, the output is `_UXXXXXXXX`, where `XXXXXXXX` is the hexadecimal code for the character.

Invalid `utf-8` substrings are escaped as `_XX'`, and if a name starts with a period, `e` or `E`, that character is escaped as `_XX`.

Variable Bounds

Specifying several upper or lower bounds on one variable is possible but **MOSEK** uses only the tightest bounds. If a variable is fixed (with `=`), then it is considered the tightest bound.

MOSEK Extensions to the LP Format

Some optimization software packages employ a more strict definition of the LP format than the one used by **MOSEK**. The limitations imposed by the strict LP format are the following:

- Quadratic terms in the constraints are not allowed.
- Names can be only 16 characters long.
- Lines must not exceed 255 characters in length.

To get around some of the inconveniences converting from other problem formats, **MOSEK** allows lines to contain 1024 characters and names may have any length (shorter than the 1024 characters).

If an LP formatted file created by **MOSEK** should satisfy the strict definition, then the parameter *MSK_IPAR_WRITE_LP_STRICT_FORMAT* should be set; note, however, that some problems cannot be written correctly as a strict LP formatted file. For instance, all names are truncated to 16 characters and hence they may lose their uniqueness and change the problem.

Internally in **MOSEK** names may contain any (printable) character, many of which cannot be used in LP names. Setting the parameters *MSK_IPAR_READ_LP_QUOTED_NAMES* and *MSK_IPAR_WRITE_LP_QUOTED_NAMES* allows **MOSEK** to use quoted names. The first parameter tells **MOSEK** to remove quotes from quoted names e.g, `"x1"`, when reading LP formatted files. The second parameter tells **MOSEK** to put quotes around any semi-illegal name (names beginning with a number or a period) and fully illegal name (containing illegal characters). As double quote is a legal character in the LP format, quoting semi-illegal names makes them legal in the pure LP format as long as they are still shorter than 16 characters. Fully illegal names are still illegal in a pure LP file.

The strict LP format

The LP format is not a formal standard and different vendors have slightly different interpretations of the LP format. To make **MOSEK**'s definition of the LP format more compatible with the definitions of other vendors set the parameter *MSK_IPAR_WRITE_LP_STRICT_FORMAT* to *"MSK_ON"*.

This setting may lead to truncation of some names and hence to an invalid LP file. The simple solution to this problem is to set the parameter *MSK_IPAR_WRITE_GENERIC_NAMES* to *"MSK_ON"* which will cause all names to be renamed systematically in the output file.

Formatting of an LP File

A few parameters control the visual formatting of LP files written by **MOSEK** in order to make it easier to read the files. These parameters are

- *MSK_IPAR_WRITE_LP_LINE_WIDTH* sets the maximum number of characters on a single line. The default value is 80 corresponding roughly to the width of a standard text document.
- *MSK_IPAR_WRITE_LP_TERMS_PER_LINE* sets the maximum number of terms per line; a term means a sign, a coefficient, and a name (for example `+ 42 elephants`). The default value is 0, meaning that there is no maximum.

Unnamed Constraints

Reading and writing an LP file with **MOSEK** may change it superficially. If an LP file contains unnamed constraints or objective these are given their generic names when the file is read (however unnamed constraints in **MOSEK** are written without names).

16.2 The MPS File Format

MOSEK supports the standard MPS format with some extensions. For a detailed description of the MPS format see the book by Nazareth *[Naz87]*.

16.2.1 MPS File Structure

The version of the MPS format supported by **MOSEK** allows specification of an optimization problem of the form

$$
\begin{array}{rccccl}
\text{maximize/minimize} & & c^T x + q_0(x) & & & \\
l^c & \leq & Ax + q(x) & \leq & u^c, & \\
l^x & \leq & x & \leq & u^x, & \\
 & & x \in \mathcal{K}, & & & \\
 & & x_{\mathcal{J}} \quad \text{integer}, & & &
\end{array} \tag{16.2}
$$

where

- $x \in \mathbb{R}^n$ is the vector of decision variables.
- $A \in \mathbb{R}^{m \times n}$ is the constraint matrix.
- $l^c \in \mathbb{R}^m$ is the lower limit on the activity for the constraints.
- $u^c \in \mathbb{R}^m$ is the upper limit on the activity for the constraints.
- $l^x \in \mathbb{R}^n$ is the lower limit on the activity for the variables.
- $u^x \in \mathbb{R}^n$ is the upper limit on the activity for the variables.
- $q : \mathbb{R}^n \rightarrow \mathbb{R}$ 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. The same applies to q_0.
- $\mathcal{K}$ is a convex cone.
- $\mathcal{J} \subseteq \{1, 2, \ldots, n\}$ is an index set of the integer-constrained variables.
- c is the vector of objective coefficients.

An MPS file with one row and one column can be illustrated like this:

```
*        1         2         3         4         5         6
*23456789012345678901234567890123456789012345678901234567890
NAME          [name]
OBJSENSE
    [objsense]
OBJNAME       [objname]
ROWS
 ?  [cname1]
COLUMNS
    [vname1]  [cname1]  [value1]       [cname2]  [value2]
RHS
    [name]    [cname1]  [value1]       [cname2]  [value2]
RANGES
    [name]    [cname1]  [value1]       [cname2]  [value2]
QSECTION      [cname1]
    [vname1]  [vname2]  [value1]       [vname3]  [value2]
QMATRIX
    [vname1]  [vname2]  [value1]
```

(continues on next page)

(continued from previous page)

```
QUADOBJ
    [vname1]  [vname2]  [value1]
QCMATRIX      [cname1]
    [vname1]  [vname2]  [value1]
BOUNDS
 ?? [name]    [vname1]  [value1]
CSECTION      [kname1]  [value1]       [ktype]
    [vname1]
ENDATA
```

Here the names in capitals are keywords of the MPS format and names in brackets are custom defined names or values. A couple of notes on the structure:

- Fields: All items surrounded by brackets appear in *fields*. The fields named "`valueN`" are numerical values. Hence, they must have the format

  ```
  [+|-]XXXXXXX.XXXXXX[[e|E][+|-]XXX]
  ```

  where

  ```
  X = [0|1|2|3|4|5|6|7|8|9].
  ```

- Sections: The MPS file consists of several sections where the names in capitals indicate the beginning of a new section. For example, `COLUMNS` denotes the beginning of the columns section.
- Comments: Lines starting with an `*` are comment lines and are ignored by **MOSEK**.
- Keys: The question marks represent keys to be specified later.
- Extensions: The sections `QSECTION` and `CSECTION` are specific **MOSEK** extensions of the MPS format. The sections `QMATRIX`, `QUADOBJ` and `QCMATRIX` are included for sake of compatibility with other vendors extensions to the MPS format.
- The standard MPS format is a fixed format, i.e. everything in the MPS file must be within certain fixed positions. **MOSEK** also supports a *free format*. See Sec. 16.2.5 for details.

Linear example `lo1.mps`

A concrete example of a MPS file is presented below:

```
* File: lo1.mps
NAME          lo1
OBJSENSE
    MAX
ROWS
 N  obj
 E  c1
 G  c2
 L  c3
COLUMNS
    x1        obj       3
    x1        c1        3
    x1        c2        2
    x2        obj       1
    x2        c1        1
    x2        c2        1
    x2        c3        2
    x3        obj       5
    x3        c1        2
    x3        c2        3
    x4        obj       1
    x4        c2        1
```

(continues on next page)

(continued from previous page)

```
    x4        c3        3
RHS
    rhs       c1        30
    rhs       c2        15
    rhs       c3        25
RANGES
BOUNDS
 UP bound     x2        10
ENDATA
```

Subsequently each individual section in the MPS format is discussed.

NAME (optional)

In this section a name ([name]) is assigned to the problem.

OBJSENSE (optional)

This is an optional section that can be used to specify the sense of the objective function. The OBJSENSE section contains one line at most which can be one of the following:

```
MIN
MINIMIZE
MAX
MAXIMIZE
```

It should be obvious what the implication is of each of these four lines.

OBJNAME (optional)

This is an optional section that can be used to specify the name of the row that is used as objective function. objname should be a valid row name.

ROWS

A record in the ROWS section has the form

```
?   [cname1]
```

where the requirements for the fields are as follows:

Field	Starting Position	Max Width	required	Description
?	2	1	Yes	Constraint key
[cname1]	5	8	Yes	Constraint name

Hence, in this section each constraint is assigned a unique name denoted by [cname1]. Please note that [cname1] starts in position 5 and the field can be at most 8 characters wide. An initial key ? must be present to specify the type of the constraint. The key can have values E, G, L, or N with the following interpretation:

Constraint type	l_i^c	u_i^c
E (equal)	finite	$= l_i^c$
G (greater)	finite	∞
L (lower)	$-\infty$	finite
N (none)	$-\infty$	∞

In the MPS format the objective vector is not specified explicitly, but one of the constraints having the key N will be used as the objective vector c . In general, if multiple N type constraints are specified, then the first will be used as the objective vector c, unless something else was specified in the section OBJNAME.

COLUMNS

In this section the elements of A are specified using one or more records having the form:

```
[vname1]  [cname1]   [value1]    [cname2]  [value2]
```

where the requirements for each field are as follows:

Field	Starting Position	Max Width	required	Description
[vname1]	5	8	Yes	Variable name
[cname1]	15	8	Yes	Constraint name
[value1]	25	12	Yes	Numerical value
[cname2]	40	8	No	Constraint name
[value2]	50	12	No	Numerical value

Hence, a record specifies one or two elements a_{ij} of A using the principle that `[vname1]` and `[cname1]` determines j and i respectively. Please note that `[cname1]` must be a constraint name specified in the `ROWS` section. Finally, `[value1]` denotes the numerical value of a_{ij} . Another optional element is specified by `[cname2]`, and `[value2]` for the variable specified by `[vname1]`. Some important comments are:

- All elements belonging to one variable must be grouped together.
- Zero elements of A should not be specified.
- At least one element for each variable should be specified.

RHS (optional)

A record in this section has the format

```
[name]    [cname1]   [value1]    [cname2]  [value2]
```

where the requirements for each field are as follows:

Field	Starting Position	Max Width	required	Description
[name]	5	8	Yes	Name of the RHS vector
[cname1]	15	8	Yes	Constraint name
[value1]	25	12	Yes	Numerical value
[cname2]	40	8	No	Constraint name
[value2]	50	12	No	Numerical value

The interpretation of a record is that `[name]` is the name of the `RHS` vector to be specified. In general, several vectors can be specified. `[cname1]` denotes a constraint name previously specified in the `ROWS` section. Now, assume that this name has been assigned to the i -h constraint and v_1 denotes the value specified by `[value1]`, then the interpretation of v_1 is:

Constraint	l_i^c	u_i^c
E	v_1	v_1
G	v_1	
L		v_1
N		

An optional second element is specified by `[cname2]` and `[value2]` and is interpreted in the same way. Please note that it is not necessary to specify zero elements, because elements are assumed to be zero.

RANGES (optional)

A record in this section has the form

```
[name]    [cname1]   [value1]    [cname2]  [value2]
```

where the requirements for each fields are as follows:

Field	Starting Position	Max Width	required	Description
[name]	5	8	Yes	Name of the RANGE vector
[cname1]	15	8	Yes	Constraint name
[value1]	25	12	Yes	Numerical value
[cname2]	40	8	No	Constraint name
[value2]	50	12	No	Numerical value

The records in this section are used to modify the bound vectors for the constraints, i.e. the values in l^c and u^c . A record has the following interpretation: [name] is the name of the RANGE vector and [cname1] is a valid constraint name. Assume that [cname1] is assigned to the i-th constraint and let v_1 be the value specified by [value1], then a record has the interpretation:

Constraint type	Sign of v_1	l_i^c	u_i^c
E	$-$	$u_i^c + v_1$	
E	$+$		$l_i^c + v_1$
G	$-$ or $+$		$l_i^c + \|v_1\|$
L	$-$ or $+$	$u_i^c - \|v_1\|$	
N			

Another constraint bound can optionally be modified using [cname2] and [value2] the same way.

QSECTION (optional)

Within the QSECTION the label [cname1] must be a constraint name previously specified in the ROWS section. The label [cname1] denotes the constraint to which the quadratic terms belong. A record in the QSECTION has the form

```
[vname1]  [vname2]    [value1]     [vname3]  [value2]
```

where the requirements for each field are:

Field	Starting Position	Max Width	required	Description
[vname1]	5	8	Yes	Variable name
[vname2]	15	8	Yes	Variable name
[value1]	25	12	Yes	Numerical value
[vname3]	40	8	No	Variable name
[value2]	50	12	No	Numerical value

A record specifies one or two elements in the lower triangular part of the Q^i matrix where [cname1] specifies the i . Hence, if the names [vname1] and [vname2] have been assigned to the k-th and j-th variable, then Q^i_{kj} is assigned the value given by [value1] An optional second element is specified in the same way by the fields [vname1], [vname3], and [value2].

The example

$$\begin{array}{lccc} \text{minimize} & -x_2 + \frac{1}{2}(2x_1^2 - 2x_1x_3 + 0.2x_2^2 + 2x_3^2) & & \\ \text{subject to} & x_1 + x_2 + x_3 & \geq & 1, \\ & x \geq 0 & & \end{array}$$

has the following MPS file representation

```
* File: qo1.mps
NAME          qo1
ROWS
 N  obj
 G  c1
COLUMNS
    x1        c1        1.0
    x2        obj       -1.0
    x2        c1        1.0
```

(continues on next page)

(continued from previous page)

```
    x3        c1        1.0
RHS
    rhs       c1        1.0
QSECTION      obj
    x1        x1        2.0
    x1        x3        -1.0
    x2        x2        0.2
    x3        x3        2.0
ENDATA
```

Regarding the QSECTIONs please note that:

- Only one QSECTION is allowed for each constraint.
- The QSECTIONs can appear in an arbitrary order after the COLUMNS section.
- All variable names occurring in the QSECTION must already be specified in the COLUMNS section.
- All entries specified in a QSECTION are assumed to belong to the lower triangular part of the quadratic term of Q .

QMATRIX/QUADOBJ (optional)

The QMATRIX and QUADOBJ sections allow to define the quadratic term of the objective function. They differ in how the quadratic term of the objective function is stored:

- QMATRIX stores all the nonzeros coefficients, without taking advantage of the symmetry of the Q matrix.
- QUADOBJ stores the upper diagonal nonzero elements of the Q matrix.

A record in both sections has the form:

```
[vname1]  [vname2]    [value1]
```

where the requirements for each field are:

Field	Starting Position	Max Width	required	Description
[vname1]	5	8	Yes	Variable name
[vname2]	15	8	Yes	Variable name
[value1]	25	12	Yes	Numerical value

A record specifies one elements of the Q matrix in the objective function . Hence, if the names [vname1] and [vname2] have been assigned to the k-th and j-th variable, then Q_{kj} is assigned the value given by [value1]. Note that a line must appear for each off-diagonal coefficient if using a QMATRIX section, while only one entry is required in a QUADOBJ section. The quadratic part of the objective function will be evaluated as $1/2x^TQx$.

The example

$$
\begin{array}{lccc}
\text{minimize} & -x_2 + \frac{1}{2}(2x_1^2 - 2x_1x_3 + 0.2x_2^2 + 2x_3^2) & & \\
\text{subject to} & x_1 + x_2 + x_3 & \geq & 1, \\
 & x \geq 0 & &
\end{array}
$$

has the following MPS file representation using QMATRIX

```
* File: qo1_matrix.mps
NAME          qo1_qmatrix
ROWS
 N  obj
 G  c1
COLUMNS
    x1        c1        1.0
    x2        obj       -1.0
```

(continues on next page)

(continued from previous page)

```
    x2        c1        1.0
    x3        c1        1.0
RHS
    rhs       c1        1.0
QMATRIX
    x1        x1        2.0
    x1        x3        -1.0
    x3        x1        -1.0
    x2        x2        0.2
    x3        x3        2.0
ENDATA
```

or the following using `QUADOBJ`

```
* File: qo1_quadobj.mps
NAME          qo1_quadobj
ROWS
 N  obj
 G  c1
COLUMNS
    x1        c1        1.0
    x2        obj       -1.0
    x2        c1        1.0
    x3        c1        1.0
RHS
    rhs       c1        1.0
QUADOBJ
    x1        x1        2.0
    x1        x3        -1.0
    x2        x2        0.2
    x3        x3        2.0
ENDATA
```

Please also note that:

- A `QMATRIX`/`QUADOBJ` section can appear in an arbitrary order after the `COLUMNS` section.
- All variable names occurring in the `QMATRIX`/`QUADOBJ` section must already be specified in the `COLUMNS` section.

`QCMATRIX` (optional)

A `QCMATRIX` section allows to specify the quadratic part of a given constraint. Within the `QCMATRIX` the label `[cname1]` must be a constraint name previously specified in the `ROWS` section. The label `[cname1]` denotes the constraint to which the quadratic term belongs. A record in the `QSECTION` has the form

```
[vname1]  [vname2]    [value1]
```

where the requirements for each field are:

Field	Starting Position	Max Width	required	Description
`[vname1]`	5	8	Yes	Variable name
`[vname2]`	15	8	Yes	Variable name
`[value1]`	25	12	Yes	Numerical value

A record specifies an entry of the Q^i matrix where `[cname1]` specifies the i. Hence, if the names `[vname1]` and `[vname2]` have been assigned to the k-th and j-th variable, then Q^i_{kj} is assigned the value given by `[value1]`. Moreover, the quadratic term is represented as $1/2x^TQx$.

The example

$$\begin{array}{lccc} \text{minimize} & x_2 & & \\ \text{subject to} & x_1 + x_2 + x_3 & \geq & 1, \\ & \frac{1}{2}(-2x_1x_3 + 0.2x_2^2 + 2x_3^2) & \leq & 10, \\ & x \geq 0 & & \end{array}$$

has the following MPS file representation

```
* File: qo1.mps
NAME          qo1
ROWS
 N  obj
 G  c1
 L  q1
COLUMNS
    x1        c1        1.0
    x2        obj       -1.0
    x2        c1        1.0
    x3        c1        1.0
RHS
    rhs       c1        1.0
    rhs       q1        10.0
QCMATRIX      q1
    x1        x1        2.0
    x1        x3        -1.0
    x3        x1        -1.0
    x2        x2        0.2
    x3        x3        2.0
ENDATA
```

Regarding the `QCMATRIX`s please note that:

- Only one `QCMATRIX` is allowed for each constraint.
- The `QCMATRIX`s can appear in an arbitrary order after the `COLUMNS` section.
- All variable names occurring in the `QSECTION` must already be specified in the `COLUMNS` section.
- `QCMATRIX` does not exploit the symmetry of Q: an off-diagonal entry (i, j) should appear twice.

`BOUNDS` (optional)

In the `BOUNDS` section changes to the default bounds vectors l^x and u^x are specified. The default bounds vectors are $l^x = 0$ and $u^x = \infty$. Moreover, it is possible to specify several sets of bound vectors. A record in this section has the form

```
?? [name]    [vname1]    [value1]
```

where the requirements for each field are:

Field	Starting Position	Max Width	Required	Description
`??`	2	2	Yes	Bound key
`[name]`	5	8	Yes	Name of the `BOUNDS` vector
`[vname1]`	15	8	Yes	Variable name
`[value1]`	25	12	No	Numerical value

Hence, a record in the `BOUNDS` section has the following interpretation: `[name]` is the name of the bound vector and `[vname1]` is the name of the variable for which the bounds are modified by the record. `??` and `[value1]` are used to modify the bound vectors according to the following table:

`??`	l_j^x	u_j^x	Made integer (added to $\mathcal{J}$)
`FR`	$-\infty$	∞	No
`FX`	v_1	v_1	No
`LO`	v_1	unchanged	No
`MI`	$-\infty$	unchanged	No
`PL`	unchanged	∞	No
`UP`	unchanged	v_1	No
`BV`	0	1	Yes
`LI`	$\lceil v_1 \rceil$	unchanged	Yes
`UI`	unchanged	$\lfloor v_1 \rfloor$	Yes

Here v_1 is the value specified by `[value1]`.

`CSECTION` (optional)

The purpose of the `CSECTION` is to specify the conic constraint

$$x \in \mathcal{K}$$

in (16.2). It is assumed that $\mathcal{K}$ satisfies the following requirements. Let

$$x^t \in \mathbb{R}^{n^t}, \quad t = 1, \ldots, k$$

be vectors comprised of parts of the decision variables x so that each decision variable is a member of exactly **one** vector x^t, for example

$$x^1 = \begin{bmatrix} x_1 \\ x_4 \\ x_7 \end{bmatrix} \quad \text{and} \quad x^2 = \begin{bmatrix} x_6 \\ x_5 \\ x_3 \\ x_2 \end{bmatrix}.$$

Next define

$$\mathcal{K} := \left\{ x \in \mathbb{R}^n : \quad x^t \in \mathcal{K}_t, \quad t = 1, \ldots, k \right\}$$

where $\mathcal{K}_t$ must have one of the following forms:

- $\mathbb{R}$ set:

$$\mathcal{K}_t = \mathbb{R}^{n^t}.$$

- Zero cone:

$$\mathcal{K}_t = \{0\} \subseteq \mathbb{R}^{n^t}. \tag{16.3}$$

- Quadratic cone:

$$\mathcal{K}_t = \left\{ x \in \mathbb{R}^{n^t} : x_1 \geq \sqrt{\sum_{j=2}^{n^t} x_j^2} \right\}. \tag{16.4}$$

- Rotated quadratic cone:

$$\mathcal{K}_t = \left\{ x \in \mathbb{R}^{n^t} : 2x_1 x_2 \geq \sum_{j=3}^{n^t} x_j^2, \quad x_1, x_2 \geq 0 \right\}. \tag{16.5}$$

- Primal exponential cone:

$$\mathcal{K}_t = \left\{ x \in \mathbb{R}^3 : x_1 \geq x_2 \exp(x_3/x_2), \quad x_1, x_2 \geq 0 \right\}. \tag{16.6}$$

- Primal power cone (with parameter $0 < \alpha < 1$):

$$\mathcal{K}_t = \left\{ x \in \mathbb{R}^{n^t} : x_1^\alpha x_2^{1-\alpha} \geq \sqrt{\sum_{j=3}^{n^t} x_j^2}, \quad x_1, x_2 \geq 0 \right\}. \tag{16.7}$$

- Dual exponential cone:

$$\mathcal{K}_t = \left\{ x \in \mathbb{R}^3 : x_1 \geq -x_3 e^{-1} \exp(x_2/x_3), \quad x_3 \leq 0, x_1 \geq 0 \right\}. \tag{16.8}$$

- Dual power cone (with parameter $0 < \alpha < 1$):

$$\mathcal{K}_t = \left\{ x \in \mathbb{R}^{n^t} : \left(\frac{x_1}{\alpha}\right)^{\alpha} \left(\frac{x_2}{1-\alpha}\right)^{1-\alpha} \geq \sqrt{\sum_{j=3}^{n^t} x_j^2}, \quad x_1, x_2 \geq 0 \right\}. \tag{16.9}$$

In general, membership in the $\mathbb{R}$ set is not specified. If a variable is not a member of any other cone then it is assumed to be a member of the $\mathbb{R}$ cone.

Next, let us study an example. Assume that the power cone

$$x_4^{1/3} x_5^{2/3} \geq |x_8|$$

and the rotated quadratic cone

$$2x_3 x_7 \geq x_1^2 + x_0^2, \quad x_3, x_7 \geq 0,$$

should be specified in the MPS file. One CSECTION is required for each cone and they are specified as follows:

```
*        1         2         3         4         5         6
*23456789012345678901234567890123456789012345678901234567890
CSECTION      konea     3e-1           PPOW
x4
x5
x8
CSECTION      koneb     0.0            RQUAD
x7
x3
x1
x0
```

In general, a CSECTION header has the format

```
CSECTION      [kname1]     [value1]     [ktype]
```

where the requirements for each field are as follows:

Field	Starting Position	Max Width	Required	Description
[kname1]	15	8	Yes	Name of the cone
[value1]	25	12	No	Cone parameter
[ktype]	40		Yes	Type of the cone.

The possible cone type keys are:

[ktype]	Members	[value1]	Interpretation.
ZERO	≥ 0	unused	Zero cone (16.3).
QUAD	≥ 1	unused	Quadratic cone (16.4).
RQUAD	≥ 2	unused	Rotated quadratic cone (16.5).
PEXP	3	unused	Primal exponential cone (16.6).
PPOW	≥ 2	α	Primal power cone (16.7).
DEXP	3	unused	Dual exponential cone (16.8).
DPOW	≥ 2	α	Dual power cone (16.9).

A record in the CSECTION has the format

```
[vname1]
```

where the requirements for each field are

Field	Starting Position	Max Width	required	Description
[vname1]	5	8	Yes	A valid variable name

A variable must occur in at most one CSECTION.

ENDATA

This keyword denotes the end of the MPS file.

16.2.2 Integer Variables

Using special bound keys in the BOUNDS section it is possible to specify that some or all of the variables should be integer-constrained i.e. be members of $\mathcal{J}$. However, an alternative method is available. This method is available only for backward compatibility and we recommend that it is not used. This method requires that markers are placed in the COLUMNS section as in the example:

```
COLUMNS
x1        obj       -10.0          c1        0.7
x1        c2        0.5            c3        1.0
x1        c4        0.1
* Start of integer-constrained variables.
MARK000   'MARKER'                 'INTORG'
x2        obj       -9.0           c1        1.0
x2        c2        0.8333333333   c3        0.66666667
x2        c4        0.25
x3        obj       1.0            c6        2.0
MARK001   'MARKER'                 'INTEND'
* End of integer-constrained variables.
```

Please note that special marker lines are used to indicate the start and the end of the integer variables. Furthermore be aware of the following

- All variables between the markers are assigned a default lower bound of 0 and a default upper bound of 1. **This may not be what is intended.** If it is not intended, the correct bounds should be defined in the BOUNDS section of the MPS formatted file.
- **MOSEK** ignores field 1, i.e. MARK0001 and MARK001, however, other optimization systems require them.
- Field 2, i.e. MARKER, must be specified including the single quotes. This implies that no row can be assigned the name MARKER.
- Field 3 is ignored and should be left blank.
- Field 4, i.e. INTORG and INTEND, must be specified.
- It is possible to specify several such integer marker sections within the COLUMNS section.

16.2.3 General Limitations

- An MPS file should be an ASCII file.

16.2.4 Interpretation of the MPS Format

Several issues related to the MPS format are not well-defined by the industry standard. However, **MOSEK** uses the following interpretation:

- If a matrix element in the COLUMNS section is specified multiple times, then the multiple entries are added together.
- If a matrix element in a QSECTION section is specified multiple times, then the multiple entries are added together.

16.2.5 The Free MPS Format

MOSEK supports a free format variation of the MPS format. The free format is similar to the MPS file format but less restrictive, e.g. it allows longer names. However, a name must not contain any blanks.

Moreover, by default a line in the MPS file must not contain more than 1024 characters. By modifying the parameter *MSK_IPAR_READ_MPS_WIDTH* an arbitrary large line width will be accepted.

The free MPS format is default. To change to the strict and other formats use the parameter *MSK_IPAR_READ_MPS_FORMAT*.

16.3 The OPF Format

The *Optimization Problem Format (OPF)* is an alternative to LP and MPS files for specifying optimization problems. It is row-oriented, inspired by the CPLEX LP format.

Apart from containing objective, constraints, bounds etc. it may contain complete or partial solutions, comments and extra information relevant for solving the problem. It is designed to be easily read and modified by hand and to be forward compatible with possible future extensions.

Intended use

The OPF file format is meant to replace several other files:

- The LP file format: Any problem that can be written as an LP file can be written as an OPF file too; furthermore it naturally accommodates ranged constraints and variables as well as arbitrary characters in names, fixed expressions in the objective, empty constraints, and conic constraints.
- Parameter files: It is possible to specify integer, double and string parameters along with the problem (or in a separate OPF file).
- Solution files: It is possible to store a full or a partial solution in an OPF file and later reload it.

16.3.1 The File Format

The format uses tags to structure data. A simple example with the basic sections may look like this:

```
[comment]
This is a comment. You may write almost anything here...
[/comment]

# This is a single-line comment.

[objective min 'myobj']
x + 3 y + x^2 + 3 y^2 + z + 1
[/objective]

[constraints]
[con 'con01'] 4 <= x + y  [/con]
[/constraints]

[bounds]
[b] -10 <= x,y <= 10  [/b]

[cone quad] x,y,z [/cone]
[/bounds]
```

A scope is opened by a tag of the form `[tag]` and closed by a tag of the form `[/tag]`. An opening tag may accept a list of unnamed and named arguments, for examples:

```
[tag value] tag with one unnamed argument [/tag]
[tag arg=value] tag with one named argument [/tag]
```

Unnamed arguments are identified by their order, while named arguments may appear in any order, but never before an unnamed argument. The `value` can be a quoted, single-quoted or double-quoted text string, i.e.

```
[tag 'value']     single-quoted value [/tag]
[tag arg='value'] single-quoted value [/tag]
[tag "value"]     double-quoted value [/tag]
[tag arg="value"] double-quoted value [/tag]
```

16.3.2 Sections

The recognized tags are

`[comment]`

A comment section. This can contain *almost* any text: Between single quotes (') or double quotes (") any text may appear. Outside quotes the markup characters ([and]) must be prefixed by backslashes. Both single and double quotes may appear alone or inside a pair of quotes if it is prefixed by a backslash.

`[objective]`

The objective function: This accepts one or two parameters, where the first one (in the above example `min`) is either `min` or `max` (regardless of case) and defines the objective sense, and the second one (above `myobj`), if present, is the objective name. The section may contain linear and quadratic expressions.

If several objectives are specified, all but the last are ignored.

`[constraints]`

This does not directly contain any data, but may contain subsections `con` defining a linear constraint.

`[con]`

Defines a single constraint; if an argument is present (`[con NAME]`) this is used as the name of the constraint, otherwise it is given a null-name. The section contains a constraint definition written as linear and quadratic expressions with a lower bound, an upper bound, with both or with an equality. Examples:

```
[constraints]
[con 'con1'] 0 <= x + y       [/con]
[con 'con2'] 0 >= x + y       [/con]
[con 'con3'] 0 <= x + y <= 10 [/con]
[con 'con4']      x + y  = 10 [/con]
[/constraints]
```

Constraint names are unique. If a constraint is specified which has the same name as a previously defined constraint, the new constraint replaces the existing one.

`[bounds]`

This does not directly contain any data, but may contain subsections `b` (linear bounds on variables) and `cone` (cones).

`[b]`

Bound definition on one or several variables separated by comma (`,`). An upper or lower bound on a variable replaces any earlier defined bound on that variable. If only one bound (upper or lower) is given only this bound is replaced. This means that upper and lower bounds can be specified separately. So the OPF bound definition:

```
[b]  x,y >= -10  [/b]
[b]  x,y <= 10   [/b]
```

results in the bound $-10 \leq x, y \leq 10$.

`[cone]`

Specifies a cone. A cone is defined as a sequence of variables which belong to a single unique cone. The supported cone types are:

- `quad`: a quadratic cone of n variables $x_1, \dots, x_n$ defines a constraint of the form

$$x_1^2 \geq \sum_{i=2}^{n} x_i^2, \quad x_1 \geq 0.$$

- `rquad`: a rotated quadratic cone of n variables $x_1, \dots, x_n$ defines a constraint of the form

$$2x_1x_2 \geq \sum_{i=3}^{n} x_i^2, \quad x_1, x_2 \geq 0.$$

- `pexp`: primal exponential cone of 3 variables x_1, x_2, x_3 defines a constraint of the form

$$x_1 \geq x_2 \exp(x_3/x_2), \quad x_1, x_2 \geq 0.$$

- `ppow` with parameter $0 < \alpha < 1$: primal power cone of n variables $x_1, \dots, x_n$ defines a constraint of the form

$$x_1^{\alpha} x_2^{1-\alpha} \geq \sqrt{\sum_{j=3}^{n} x_j^2}, \quad x_1, x_2 \geq 0.$$

- `dexp`: dual exponential cone of 3 variables x_1, x_2, x_3 defines a constraint of the form

$$x_1 \geq -x_3 e^{-1} \exp(x_2/x_3), \quad x_3 \leq 0, x_1 \geq 0.$$

- `dpow` with parameter $0 < \alpha < 1$: dual power cone of n variables $x_1, \dots, x_n$ defines a constraint of the form

$$\left(\frac{x_1}{\alpha}\right)^{\alpha} \left(\frac{x_2}{1-\alpha}\right)^{1-\alpha} \geq \sqrt{\sum_{j=3}^{n^t} x_j^2}, \quad x_1, x_2 \geq 0.$$

- `zero`: zero cone of n variables $x_1, \dots, x_n$ defines a constraint of the form

$$x_1 = \cdots = x_n = 0$$

A `[bounds]`-section example:

```
[bounds]
[b]  0 <= x,y <= 10  [/b] # ranged bound
[b] 10 >= x,y >=  0  [/b] # ranged bound
[b]  0 <= x,y <= inf [/b] # using inf
[b]       x,y free   [/b] # free variables
# Let (x,y,z,w) belong to the cone K
[cone rquad] x,y,z,w  [/cone] # rotated quadratic cone
[cone ppow '3e-01' 'a'] x1, x2, x3 [/cone] # power cone with alpha=1/3 and name 'a'
[/bounds]
```

By default all variables are free.

`[variables]`

This defines an ordering of variables as they should appear in the problem. This is simply a space-separated list of variable names.

[integer]

This contains a space-separated list of variables and defines the constraint that the listed variables must be integer-valued.

[hints]

This may contain only non-essential data; for example estimates of the number of variables, constraints and non-zeros. Placed before all other sections containing data this may reduce the time spent reading the file.

In the `hints` section, any subsection which is not recognized by **MOSEK** is simply ignored. In this section a hint is defined as follows:

```
[hint ITEM] value [/hint]
```

The hints recognized by **MOSEK** are:

- `numvar` (number of variables),
- `numcon` (number of linear/quadratic constraints),
- `numanz` (number of linear non-zeros in constraints),
- `numqnz` (number of quadratic non-zeros in constraints).

[solutions]

This section can contain a set of full or partial solutions to a problem. Each solution must be specified using a `[solution]`-section, i.e.

```
[solutions]
[solution]...[/solution] #solution 1
[solution]...[/solution] #solution 2
#other solutions....
[solution]...[/solution] #solution n
[/solutions]
```

The syntax of a `[solution]`-section is the following:

```
[solution SOLTYPE status=STATUS]...[/solution]
```

where `SOLTYPE` is one of the strings

- `interior`, a non-basic solution,
- `basic`, a basic solution,
- `integer`, an integer solution,

and `STATUS` is one of the strings

- `UNKNOWN`,
- `OPTIMAL`,
- `INTEGER_OPTIMAL`,
- `PRIM_FEAS`,
- `DUAL_FEAS`,
- `PRIM_AND_DUAL_FEAS`,
- `NEAR_OPTIMAL`,
- `NEAR_PRIM_FEAS`,

- NEAR_DUAL_FEAS,
- NEAR_PRIM_AND_DUAL_FEAS,
- PRIM_INFEAS_CER,
- DUAL_INFEAS_CER,
- NEAR_PRIM_INFEAS_CER,
- NEAR_DUAL_INFEAS_CER,
- NEAR_INTEGER_OPTIMAL.

Most of these values are irrelevant for input solutions; when constructing a solution for simplex hot-start or an initial solution for a mixed integer problem the safe setting is UNKNOWN.

A [solution]-section contains [con] and [var] sections. Each [con] and [var] section defines solution information for a single variable or constraint, specified as list of KEYWORD/value pairs, in any order, written as

```
KEYWORD=value
```

Allowed keywords are as follows:

- sk. The status of the item, where the value is one of the following strings:
  - LOW, the item is on its lower bound.
  - UPR, the item is on its upper bound.
  - FIX, it is a fixed item.
  - BAS, the item is in the basis.
  - SUPBAS, the item is super basic.
  - UNK, the status is unknown.
  - INF, the item is outside its bounds (infeasible).
- lvl Defines the level of the item.
- sl Defines the level of the dual variable associated with its lower bound.
- su Defines the level of the dual variable associated with its upper bound.
- sn Defines the level of the variable associated with its cone.
- y Defines the level of the corresponding dual variable (for constraints only).

A [var] section should always contain the items sk, lvl, sl and su. Items sl and su are not required for integer solutions.

A [con] section should always contain sk, lvl, sl, su and y.

An example of a solution section

```
[solution basic status=UNKNOWN]
[var x0] sk=LOW    lvl=5.0      [/var]
[var x1] sk=UPR    lvl=10.0     [/var]
[var x2] sk=SUPBAS lvl=2.0  sl=1.5 su=0.0 [/var]

[con c0] sk=LOW    lvl=3.0 y=0.0 [/con]
[con c0] sk=UPR    lvl=0.0 y=5.0 [/con]
[/solution]
```

- [vendor] This contains solver/vendor specific data. It accepts one argument, which is a vendor ID – for **MOSEK** the ID is simply mosek – and the section contains the subsection parameters defining solver parameters. When reading a vendor section, any unknown vendor can be safely ignored. This is described later.

Comments using the # may appear anywhere in the file. Between the # and the following line-break any text may be written, including markup characters.

16.3.3 Numbers

Numbers, when used for parameter values or coefficients, are written in the usual way by the `printf` function. That is, they may be prefixed by a sign (`+` or `-`) and may contain an integer part, decimal part and an exponent. The decimal point is always `.` (a dot). Some examples are

```
1
1.0
.0
1.
1e10
1e+10
1e-10
```

Some *invalid* examples are

```
e10   # invalid, must contain either integer or decimal part
.     # invalid
.e10  # invalid
```

More formally, the following standard regular expression describes numbers as used:

```
[+|-]?([0-9]+[.][0-9]*|[.][0-9]+)([eE][+|-]?[0-9]+)?
```

16.3.4 Names

Variable names, constraint names and objective name may contain arbitrary characters, which in some cases must be enclosed by quotes (single or double) that in turn must be preceded by a backslash. Unquoted names must begin with a letter (`a-z` or `A-Z`) and contain only the following characters: the letters `a-z` and `A-Z`, the digits `0-9`, braces (`{` and `}`) and underscore (`_`).

Some examples of legal names:

```
an_unquoted_name
another_name{123}
'single quoted name'
"double quoted name"
"name with \\"quote\\" in it"
"name with []s in it"
```

16.3.5 Parameters Section

In the `vendor` section solver parameters are defined inside the `parameters` subsection. Each parameter is written as

```
[p PARAMETER_NAME] value [/p]
```

where `PARAMETER_NAME` is replaced by a **MOSEK** parameter name, usually of the form `MSK_IPAR_...`, `MSK_DPAR_...` or `MSK_SPAR_...`, and the `value` is replaced by the value of that parameter; both integer values and named values may be used. Some simple examples are

```
[vendor mosek]
[parameters]
[p MSK_IPAR_OPF_MAX_TERMS_PER_LINE] 10     [/p]
[p MSK_IPAR_OPF_WRITE_PARAMETERS]   MSK_ON [/p]
[p MSK_DPAR_DATA_TOL_BOUND_INF]     1.0e18 [/p]
[/parameters]
[/vendor]
```

16.3.6 Writing OPF Files from MOSEK

To write an OPF file then make sure the file extension is .opf.

Then modify the following parameters to define what the file should contain:

MSK_IPAR_OPF_WRITE_SOL_BAS	Include basic solution, if defined.
MSK_IPAR_OPF_WRITE_SOL_ITG	Include integer solution, if defined.
MSK_IPAR_OPF_WRITE_SOL_ITR	Include interior solution, if defined.
MSK_IPAR_OPF_WRITE_SOLUTIONS	Include solutions if they are defined. If this is off, no solutions are included.
MSK_IPAR_OPF_WRITE_HEADER	Include a small header with comments.
MSK_IPAR_OPF_WRITE_PROBLEM	Include the problem itself — objective, constraints and bounds.
MSK_IPAR_OPF_WRITE_PARAMETERS	Include all parameter settings.
MSK_IPAR_OPF_WRITE_HINTS	Include hints about the size of the problem.

16.3.7 Examples

This section contains a set of small examples written in OPF and describing how to formulate linear, quadratic and conic problems.

Linear Example lo1.opf

Consider the example:

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

having the bounds

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

In the OPF format the example is displayed as shown in Listing 16.1.

Listing 16.1: Example of an OPF file for a linear problem.

```
[comment]
  The lo1 example in OPF format
[/comment]

[hints]
  [hint NUMVAR] 4 [/hint]
  [hint NUMCON] 3 [/hint]
  [hint NUMANZ] 9 [/hint]
[/hints]

[variables disallow_new_variables]
  x1 x2 x3 x4
[/variables]

[objective maximize 'obj']
   3 x1 + x2 + 5 x3 + x4
[/objective]

[constraints]
  [con 'c1'] 3 x1 +    x2 + 2 x3          = 30 [/con]
  [con 'c2'] 2 x1 +    x2 + 3 x3 +    x4 >= 15 [/con]
  [con 'c3']         2 x2         + 3 x4 <= 25 [/con]
[/constraints]

[bounds]
```

(continues on next page)

(continued from previous page)

```
  [b] 0 <= * [/b]
  [b] 0 <= x2 <= 10 [/b]
[/bounds]
```

Quadratic Example qo1.opf

An example of a quadratic optimization problem is

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

This can be formulated in `opf` as shown below.

Listing 16.2: Example of an OPF file for a quadratic problem.

```
[comment]
  The qo1 example in OPF format
[/comment]

[hints]
  [hint NUMVAR] 3 [/hint]
  [hint NUMCON] 1 [/hint]
  [hint NUMANZ] 3 [/hint]
  [hint NUMQNZ] 4 [/hint]
[/hints]

[variables disallow_new_variables]
  x1 x2 x3
[/variables]

[objective minimize 'obj']
  # The quadratic terms are often written with a factor of 1/2 as here,
  # but this is not required.

   - x2 + 0.5 ( 2.0 x1 ^ 2 - 2.0 x3 * x1 + 0.2 x2 ^ 2 + 2.0 x3 ^ 2 )
[/objective]

[constraints]
  [con 'c1'] 1.0 <= x1 + x2 + x3 [/con]
[/constraints]

[bounds]
  [b] 0 <= * [/b]
[/bounds]
```

Conic Quadratic Example cqo1.opf

Consider the example:

$$\begin{array}{lccc} \text{minimize} & x_3 + x_4 + x_5 & & \\ \text{subject to} & x_0 + x_1 + 2x_2 & = & 1, \\ & x_0, x_1, x_2 & \geq & 0, \\ & x_3 \geq \sqrt{x_0^2 + x_1^2}, & & \\ & 2x_4x_5 \geq x_2^2. & & \end{array}$$

Please note that the type of the cones is defined by the parameter to `[cone ...]`; the content of the `cone`-section is the names of variables that belong to the cone. The resulting OPF file is in Listing 16.3.

Listing 16.3: Example of an OPF file for a conic quadratic problem.

```
[comment]
  The cqo1 example in OPF format.
[/comment]

[hints]
  [hint NUMVAR] 6 [/hint]
  [hint NUMCON] 1 [/hint]
  [hint NUMANZ] 3 [/hint]
[/hints]

[variables disallow_new_variables]
  x1 x2 x3 x4 x5 x6
[/variables]

[objective minimize 'obj']
   x4 + x5 + x6
[/objective]

[constraints]
  [con 'c1']  x1 + x2 + 2e+00 x3 = 1e+00 [/con]
[/constraints]

[bounds]
  # We let all variables default to the positive orthant
  [b] 0 <= * [/b]

  # ...and change those that differ from the default
  [b] x4,x5,x6 free [/b]

  # Define quadratic cone: x4 >= sqrt( x1^2 + x2^2 )
  [cone quad 'k1'] x4, x1, x2 [/cone]

  # Define rotated quadratic cone: 2 x5 x6 >= x3^2
  [cone rquad 'k2'] x5, x6, x3 [/cone]
[/bounds]
```

Mixed Integer Example `milo1.opf`

Consider the mixed integer problem:

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

This can be implemented in `OPF` with the file in Listing 16.4.

Listing 16.4: Example of an OPF file for a mixed-integer linear problem.

```
[comment]
  The milo1 example in OPF format
[/comment]

[hints]
  [hint NUMVAR] 2 [/hint]
  [hint NUMCON] 2 [/hint]
  [hint NUMANZ] 4 [/hint]
[/hints]
```

(continues on next page)

(continued from previous page)

```
[variables disallow_new_variables]
  x1 x2
[/variables]

[objective maximize 'obj']
   x1 + 6.4e-1 x2
[/objective]

[constraints]
  [con 'c1'] 5e+1 x1 + 3.1e+1 x2 <= 2.5e+2 [/con]
  [con 'c2'] -4 <= 3 x1 - 2 x2 [/con]
[/constraints]

[bounds]
  [b] 0 <= * [/b]
[/bounds]

[integer]
  x1 x2
[/integer]
```

16.4 The CBF Format

This document constitutes the technical reference manual of the *Conic Benchmark Format* with file extension: `.cbf` or `.CBF`. It unifies linear, second-order cone (also known as conic quadratic) and semidefinite optimization with mixed-integer variables. The format has been designed with benchmark libraries in mind, and therefore focuses on compact and easily parsable representations. The problem structure is separated from the problem data, and the format moreover facilitates benchmarking of hotstart capability through sequences of changes.

16.4.1 How Instances Are Specified

This section defines the spectrum of conic optimization problems that can be formulated in terms of the keywords of the CBF format.

In the CBF format, conic optimization problems are considered in the following form:

$$\begin{array}{lll} \min / \max & g^{obj} & \\ & g_i \in \mathcal{K}_i, & i \in \mathcal{I}, \\ \text{s.t.} & G_i \in \mathcal{K}_i, & i \in \mathcal{I}^{PSD}, \\ & x_j \in \mathcal{K}_j, & j \in \mathcal{J}, \\ & \overline{X}_j \in \mathcal{K}_j, & j \in \mathcal{J}^{PSD}. \end{array} \tag{16.10}$$

- **Variables** are either scalar variables, x_j for $j \in \mathcal{J}$, or variables, $\overline{X}_j$ for $j \in \mathcal{J}^{PSD}$. Scalar variables can also be declared as integer.
- **Constraints** are affine expressions of the variables, either scalar-valued g_i for $i \in \mathcal{I}$, or matrix-valued G_i for $i \in \mathcal{I}^{PSD}$

$$g_i = \sum_{j \in \mathcal{J}^{PSD}} \langle F_{ij}, X_j \rangle + \sum_{j \in \mathcal{J}} a_{ij} x_j + b_i,$$

$$G_i = \sum_{j \in \mathcal{J}} x_j H_{ij} + D_i.$$

- The **objective function** is a scalar-valued affine expression of the variables, either to be minimized or maximized. We refer to this expression as g^{obj}

$$g^{obj} = \sum_{j \in \mathcal{J}^{PSD}} \langle F_j^{obj}, X_j \rangle + \sum_{j \in \mathcal{J}} a_j^{obj} x_j + b^{obj}.$$

CBF format can represent the following cones $\mathcal{K}$:

- **Free domain** - A cone in the linear family defined by

$$\{x \in \mathbb{R}^n\}, \text{ for } n \geq 1.$$

- **Positive orthant** - A cone in the linear family defined by

$$\{x \in \mathbb{R}^n \mid x_j \geq 0 \text{ for } j = 1, \ldots, n\}, \text{ for } n \geq 1.$$

- **Negative orthant** - A cone in the linear family defined by

$$\{x \in \mathbb{R}^n \mid x_j \leq 0 \text{ for } j = 1, \ldots, n\}, \text{ for } n \geq 1.$$

- **Fixpoint zero** - A cone in the linear family defined by

$$\{x \in \mathbb{R}^n \mid x_j = 0 \text{ for } j = 1, \ldots, n\}, \text{ for } n \geq 1.$$

- **Quadratic cone** - A cone in the second-order cone family defined by

$$\left\{ \begin{pmatrix} p \\ x \end{pmatrix} \in \mathbb{R} \times \mathbb{R}^{n-1},\ p^2 \geq x^T x,\ p \geq 0 \right\}, \text{ for } n \geq 2.$$

- **Rotated quadratic cone** - A cone in the second-order cone family defined by

$$\left\{ \begin{pmatrix} p \\ q \\ x \end{pmatrix} \in \mathbb{R} \times \mathbb{R} \times \mathbb{R}^{n-2},\ 2pq \geq x^T x,\ p \geq 0,\ q \geq 0 \right\}, \text{ for } n \geq 3.$$

16.4.2 The Structure of CBF Files

This section defines how information is written in the CBF format, without being specific about the type of information being communicated.

All information items belong to exactly one of the three groups of information. These information groups, and the order they must appear in, are:

1. File format.
2. Problem structure.
3. Problem data.

The first group, file format, provides information on how to interpret the file. The second group, problem structure, provides the information needed to deduce the type and size of the problem instance. Finally, the third group, problem data, specifies the coefficients and constants of the problem instance.

Information items

The format is composed as a list of information items. The first line of an information item is the `KEYWORD`, revealing the type of information provided. The second line - of some keywords only - is the `HEADER`, typically revealing the size of information that follows. The remaining lines are the `BODY` holding the actual information to be specified.

```
KEYWORD
BODY

KEYWORD
HEADER
BODY
```

The KEYWORD determines how each line in the HEADER and BODY is structured. Moreover, the number of lines in the BODY follows either from the KEYWORD, the HEADER, or from another information item required to precede it.

Embedded hotstart-sequences

A sequence of problem instances, based on the same problem structure, is within a single file. This is facilitated via the CHANGE within the problem data information group, as a separator between the information items of each instance. The information items following a CHANGE keyword are appending to, or changing (e.g., setting coefficients back to their default value of zero), the problem data of the preceding instance.

The sequence is intended for benchmarking of hotstart capability, where the solvers can reuse their internal state and solution (subject to the achieved accuracy) as warmpoint for the succeeding instance. Whenever this feature is unsupported or undesired, the keyword CHANGE should be interpreted as the end of file.

File encoding and line width restrictions

The format is based on the US-ASCII printable character set with two extensions as listed below. Note, by definition, that none of these extensions can be misinterpreted as printable US-ASCII characters:

- A line feed marks the end of a line, carriage returns are ignored.
- Comment-lines may contain unicode characters in UTF-8 encoding.

The line width is restricted to 512 bytes, with 3 bytes reserved for the potential carriage return, line feed and null-terminator.

Integers and floating point numbers must follow the ISO C decimal string representation in the standard C locale. The format does not impose restrictions on the magnitude of, or number of significant digits in numeric data, but the use of 64-bit integers and 64-bit IEEE 754 floating point numbers should be sufficient to avoid loss of precision.

Comment-line and whitespace rules

The format allows single-line comments respecting the following rule:

- Lines having first byte equal to '#' (US-ASCII 35) are comments, and should be ignored. Comments are only allowed between information items.

Given that a line is not a comment-line, whitespace characters should be handled according to the following rules:

- Leading and trailing whitespace characters should be ignored.
  - The seperator between multiple pieces of information on one line, is either one or more whitespace characters.
- Lines containing only whitespace characters are empty, and should be ignored. Empty lines are only allowed between information items.

16.4.3 Problem Specification

The problem structure

The problem structure defines the objective sense, whether it is minimization and maximization. It also defines the index sets, $\mathcal{J}$, $\mathcal{J}^{PSD}$, $\mathcal{I}$ and $\mathcal{I}^{PSD}$, which are all numbered from zero, $\{0, 1, \ldots\}$, and empty until explicitly constructed.

- **Scalar variables** are constructed in vectors restricted to a conic domain, such as $(x_0, x_1) \in \mathbb{R}_+^2$, $(x_2, x_3, x_4) \in \mathcal{Q}^3$, etc. In terms of the Cartesian product, this generalizes to

  $$x \in \mathcal{K}_1^{n_1} \times \mathcal{K}_2^{n_2} \times \cdots \times \mathcal{K}_k^{n_k}$$

  which in the CBF format becomes:

  ```
  VAR
  n k
  K1 n1
  K2 n2
  ...
  Kk nk
  ```

  where $\sum_i n_i = n$ is the total number of scalar variables. The list of supported cones is found in Table 16.3. Integrality of scalar variables can be specified afterwards.

- **PSD variables** are constructed one-by-one. That is, $X_j \succeq \mathbf{0}^{n_j \times n_j}$ for $j \in \mathcal{J}^{PSD}$, constructs a matrix-valued variable of size $n_j \times n_j$ restricted to be symmetric positive semidefinite. In the CBF format, this list of constructions becomes:

  ```
  PSDVAR
  N
  n1
  n2
  ...
  nN
  ```

  where N is the total number of PSD variables.

- **Scalar constraints** are constructed in vectors restricted to a conic domain, such as $(g_0, g_1) \in \mathbb{R}_+^2$, $(g_2, g_3, g_4) \in \mathcal{Q}^3$, etc. In terms of the Cartesian product, this generalizes to

  $$g \in \mathcal{K}_1^{m_1} \times \mathcal{K}_2^{m_2} \times \cdots \times \mathcal{K}_k^{m_k}$$

  which in the CBF format becomes:

  ```
  CON
  m k
  K1 m1
  K2 m2
  ..
  Kk mk
  ```

  where $\sum_i m_i = m$ is the total number of scalar constraints. The list of supported cones is found in Table 16.3.

- **PSD constraints** are constructed one-by-one. That is, $G_i \succeq \mathbf{0}^{m_i \times m_i}$ for $i \in \mathcal{I}^{PSD}$, constructs a matrix-valued affine expressions of size $m_i \times m_i$ restricted to be symmetric positive semidefinite. In the CBF format, this list of constructions becomes

  ```
  PSDCON
  M
  m1
  m2
  ..
  mM
  ```

where M is the total number of PSD constraints.

With the objective sense, variables (with integer indications) and constraints, the definitions of the many affine expressions follow in problem data.

Problem data

The problem data defines the coefficients and constants of the affine expressions of the problem instance. These are considered zero until explicitly defined, implying that instances with no keywords from this information group are, in fact, valid. Duplicating or conflicting information is a failure to comply with the standard. Consequently, two coefficients written to the same position in a matrix (or to transposed positions in a symmetric matrix) is an error.

The affine expressions of the objective, g^{obj}, of the scalar constraints, g_i, and of the PSD constraints, G_i, are defined separately. The following notation uses the standard trace inner product for matrices, $\langle X, Y\rangle = \sum_{i,j} X_{ij}Y_{ij}$.

- The affine expression of the objective is defined as

$$g^{obj} = \sum_{j\in\mathcal{J}^{PSD}} \langle F_j^{obj}, X_j\rangle + \sum_{j\in\mathcal{J}} a_j^{obj} x_j + b^{obj},$$

  in terms of the symmetric matrices, F_j^{obj}, and scalars, a_j^{obj} and b^{obj}.

- The affine expressions of the scalar constraints are defined, for $i \in \mathcal{I}$, as

$$g_i = \sum_{j\in\mathcal{J}^{PSD}} \langle F_{ij}, X_j\rangle + \sum_{j\in\mathcal{J}} a_{ij} x_j + b_i,$$

  in terms of the symmetric matrices, F_{ij}, and scalars, a_{ij} and b_i.

- The affine expressions of the PSD constraints are defined, for $i \in \mathcal{I}^{PSD}$, as

$$G_i = \sum_{j\in\mathcal{J}} x_j H_{ij} + D_i,$$

  in terms of the symmetric matrices, H_{ij} and D_i.

List of cones

The format uses an explicit syntax for symmetric positive semidefinite cones as shown above. For scalar variables and constraints, constructed in vectors, the supported conic domains and their minimum sizes are given as follows.

Table 16.3: Cones available in the CBF format

Name	CBF keyword	Cone family
Free domain	`F`	linear
Positive orthant	`L+`	linear
Negative orthant	`L-`	linear
Fixpoint zero	`L=`	linear
Quadratic cone	`Q`	second-order
Rotated quadratic cone	`QR`	second-order

16.4.4 File Format Keywords

VER

Description: The version of the Conic Benchmark Format used to write the file.

`HEADER`: None

`BODY`: One line formatted as:

```
INT
```

This is the version number.
Must appear exactly once in a file, as the first keyword.

OBJSENSE

Description: Define the objective sense.
`HEADER`: None
`BODY`: One line formatted as:

```
STR
```

having `MIN` indicates minimize, and `MAX` indicates maximize. Capital letters are required.
Must appear exactly once in a file.

PSDVAR

Description: Construct the PSD variables.
`HEADER`: One line formatted as:

```
INT
```

This is the number of PSD variables in the problem.
`BODY`: A list of lines formatted as:

```
INT
```

This indicates the number of rows (equal to the number of columns) in the matrix-valued PSD variable. The number of lines should match the number stated in the header.

VAR

Description: Construct the scalar variables.
`HEADER`: One line formatted as:

```
INT INT
```

This is the number of scalar variables, followed by the number of conic domains they are restricted to.
`BODY`: A list of lines formatted as:

```
STR INT
```

This indicates the cone name (see Table 16.3), and the number of scalar variables restricted to this cone. These numbers should add up to the number of scalar variables stated first in the header. The number of lines should match the second number stated in the header.

INT

Description: Declare integer requirements on a selected subset of scalar variables.
`HEADER`: one line formatted as:

```
INT
```

This is the number of integer scalar variables in the problem.
`BODY`: a list of lines formatted as:

```
INT
```

This indicates the scalar variable index $j \in \mathcal{J}$. The number of lines should match the number stated in the header.
Can only be used after the keyword `VAR`.

PSDCON

Description: Construct the PSD constraints.

`HEADER`: One line formatted as:

```
INT
```

This is the number of PSD constraints in the problem.

`BODY`: A list of lines formatted as:

```
INT
```

This indicates the number of rows (equal to the number of columns) in the matrix-valued affine expression of the PSD constraint. The number of lines should match the number stated in the header.

Can only be used after these keywords: `PSDVAR`, `VAR`.

CON

Description: Construct the scalar constraints.

`HEADER`: One line formatted as:

```
INT INT
```

This is the number of scalar constraints, followed by the number of conic domains they restrict to.

`BODY`: A list of lines formatted as:

```
STR INT
```

This indicates the cone name (see Table 16.3), and the number of affine expressions restricted to this cone. These numbers should add up to the number of scalar constraints stated first in the header. The number of lines should match the second number stated in the header.

Can only be used after these keywords: `PSDVAR`, `VAR`

OBJFCOORD

Description: Input sparse coordinates (quadruplets) to define the symmetric matrices F_j^{obj}, as used in the objective.

`HEADER`: One line formatted as:

```
INT
```

This is the number of coordinates to be specified.

`BODY`: A list of lines formatted as:

```
INT INT INT REAL
```

This indicates the PSD variable index $j \in \mathcal{J}^{PSD}$, the row index, the column index and the coefficient value. The number of lines should match the number stated in the header.

OBJACOORD

Description: Input sparse coordinates (pairs) to define the scalars, a_j^{obj}, as used in the objective.

`HEADER`: One line formatted as:

```
INT
```

This is the number of coordinates to be specified.

`BODY`: A list of lines formatted as:

```
INT REAL
```

This indicates the scalar variable index $j \in \mathcal{J}$ and the coefficient value. The number of lines should match the number stated in the header.

OBJBCOORD

Description: Input the scalar, b^{obj}, as used in the objective.
`HEADER`: None.
`BODY`: One line formatted as:

```
REAL
```

This indicates the coefficient value.

FCOORD

Description: Input sparse coordinates (quintuplets) to define the symmetric matrices, F_{ij}, as used in the scalar constraints.
`HEADER`: One line formatted as:

```
INT
```

This is the number of coordinates to be specified.
`BODY`: A list of lines formatted as:

```
INT INT INT INT REAL
```

This indicates the scalar constraint index $i \in \mathcal{I}$, the PSD variable index $j \in \mathcal{J}^{PSD}$, the row index, the column index and the coefficient value. The number of lines should match the number stated in the header.

ACOORD

Description: Input sparse coordinates (triplets) to define the scalars, a_{ij}, as used in the scalar constraints.
`HEADER`: One line formatted as:

```
INT
```

This is the number of coordinates to be specified.
`BODY`: A list of lines formatted as:

```
INT INT REAL
```

This indicates the scalar constraint index $i \in \mathcal{I}$, the scalar variable index $j \in \mathcal{J}$ and the coefficient value. The number of lines should match the number stated in the header.

BCOORD

Description: Input sparse coordinates (pairs) to define the scalars, b_i, as used in the scalar constraints.
`HEADER`: One line formatted as:

```
INT
```

This is the number of coordinates to be specified.
`BODY`: A list of lines formatted as:

```
INT REAL
```

This indicates the scalar constraint index $i \in \mathcal{I}$ and the coefficient value. The number of lines should match the number stated in the header.

HCOORD

Description: Input sparse coordinates (quintuplets) to define the symmetric matrices, H_{ij}, as used in the PSD constraints.
`HEADER`: One line formatted as:

```
INT
```

This is the number of coordinates to be specified.

BODY: A list of lines formatted as

```
INT INT INT INT REAL
```

This indicates the PSD constraint index $i \in \mathcal{I}^{PSD}$, the scalar variable index $j \in \mathcal{J}$, the row index, the column index and the coefficient value. The number of lines should match the number stated in the header.

DCOORD

Description: Input sparse coordinates (quadruplets) to define the symmetric matrices, D_i, as used in the PSD constraints.

HEADER: One line formatted as

```
INT
```

This is the number of coordinates to be specified.

BODY: A list of lines formatted as:

```
INT INT INT REAL
```

This indicates the PSD constraint index $i \in \mathcal{I}^{PSD}$, the row index, the column index and the coefficient value. The number of lines should match the number stated in the header.

CHANGE

Start of a new instance specification based on changes to the previous. Can be interpreted as the end of file when the hotstart-sequence is unsupported or undesired.

BODY: None

Header: None

16.4.5 CBF Format Examples

Minimal Working Example

The conic optimization problem (16.11) , has three variables in a quadratic cone - first one is integer - and an affine expression in domain 0 (equality constraint).

$$\begin{array}{ll} \text{minimize} & 5.1\,x_0 \\ \text{subject to} & 6.2\,x_1 + 7.3\,x_2 - 8.4 \in \{0\} \\ & x \in \mathcal{Q}^3,\ x_0 \in \mathbb{Z}. \end{array} \tag{16.11}$$

Its formulation in the Conic Benchmark Format begins with the version of the CBF format used, to safeguard against later revisions.

```
VER
1
```

Next follows the problem structure, consisting of the objective sense, the number and domain of variables, the indices of integer variables, and the number and domain of scalar-valued affine expressions (i.e., the equality constraint).

```
OBJSENSE
MIN

VAR
3 1
Q 3

INT
1
0
```

(continues on next page)

(continued from previous page)

```
CON
1 1
L= 1
```

Finally follows the problem data, consisting of the coefficients of the objective, the coefficients of the constraints, and the constant terms of the constraints. All data is specified on a sparse coordinate form.

```
OBJACOORD
1
0 5.1

ACOORD
2
0 1 6.2
0 2 7.3

BCOORD
1
0 -8.4
```

This concludes the example.

Mixing Linear, Second-order and Semidefinite Cones

The conic optimization problem (16.12), has a semidefinite cone, a quadratic cone over unordered subindices, and two equality constraints.

$$
\begin{array}{lrcl}
\text{minimize} & \left\langle \begin{bmatrix} 2 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 2 \end{bmatrix}, X_1 \right\rangle + x_1 & & \\
\text{subject to} & \left\langle \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, X_1 \right\rangle + x_1 & = & 1.0\,, \\
& \left\langle \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, X_1 \right\rangle + x_0 + x_2 & = & 0.5\,, \\
& x_1 \geq \sqrt{x_0^2 + x_2^2}\,, & & \\
& X_1 \succeq \mathbf{0}\,. & &
\end{array}
\tag{16.12}
$$

The equality constraints are easily rewritten to the conic form, $(g_0, g_1) \in \{0\}^2$, by moving constants such that the right-hand-side becomes zero. The quadratic cone does not fit under the `VAR` keyword in this variable permutation. Instead, it takes a scalar constraint $(g_2, g_3, g_4) = (x_1, x_0, x_2) \in \mathcal{Q}^3$, with scalar variables constructed as $(x_0, x_1, x_2) \in \mathbb{R}^3$. Its formulation in the CBF format is reported in the following list

```
# File written using this version of the Conic Benchmark Format:
#     | Version 1.
VER
1

# The sense of the objective is:
#     | Minimize.
OBJSENSE
MIN

# One PSD variable of this size:
#     | Three times three.
PSDVAR
1
3
```

(continues on next page)

(continued from previous page)

```
# Three scalar variables in this one conic domain:
#     | Three are free.
VAR
3 1
F 3

# Five scalar constraints with affine expressions in two conic domains:
#     | Two are fixed to zero.
#     | Three are in conic quadratic domain.
CON
5 2
L= 2
Q 3

# Five coordinates in F^{obj}_j coefficients:
#     | F^{obj}[0][0,0] = 2.0
#     | F^{obj}[0][1,0] = 1.0
#     | and more...
OBJFCOORD
5
0 0 0 2.0
0 1 0 1.0
0 1 1 2.0
0 2 1 1.0
0 2 2 2.0

# One coordinate in a^{obj}_j coefficients:
#     | a^{obj}[1] = 1.0
OBJACOORD
1
1 1.0

# Nine coordinates in F_ij coefficients:
#     | F[0,0][0,0] = 1.0
#     | F[0,0][1,1] = 1.0
#     | and more...
FCOORD
9
0 0 0 0 1.0
0 0 1 1 1.0
0 0 2 2 1.0
1 0 0 0 1.0
1 0 1 0 1.0
1 0 2 0 1.0
1 0 1 1 1.0
1 0 2 1 1.0
1 0 2 2 1.0

# Six coordinates in a_ij coefficients:
#     | a[0,1] = 1.0
#     | a[1,0] = 1.0
#     | and more...
ACOORD
6
0 1 1.0
1 0 1.0
1 2 1.0
2 1 1.0
3 0 1.0
4 2 1.0
```

(continues on next page)

(continued from previous page)

```
# Two coordinates in b_i coefficients:
#     | b[0] = -1.0
#     | b[1] = -0.5
BCOORD
2
0 -1.0
1 -0.5
```

Mixing Semidefinite Variables and Linear Matrix Inequalities

The standard forms in semidefinite optimization are usually based either on semidefinite variables or linear matrix inequalities. In the CBF format, both forms are supported and can even be mixed as shown in.

$$
\begin{array}{lllll}
\text{minimize} & \left\langle \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, X_1 \right\rangle + x_1 + x_2 + 1 & & \\
\text{subject to} & \left\langle \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, X_1 \right\rangle - x_1 - x_2 & \geq & 0.0\,, \\
& x_1 \begin{bmatrix} 0 & 1 \\ 1 & 3 \end{bmatrix} + x_2 \begin{bmatrix} 3 & 1 \\ 1 & 0 \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \succeq & \mathbf{0}\,, \\
& X_1 \succeq \mathbf{0}\,. & &
\end{array}
\tag{16.13}
$$

Its formulation in the CBF format is written in what follows

```
# File written using this version of the Conic Benchmark Format:
#     | Version 1.
VER
1

# The sense of the objective is:
#     | Minimize.
OBJSENSE
MIN

# One PSD variable of this size:
#     | Two times two.
PSDVAR
1
2

# Two scalar variables in this one conic domain:
#     | Two are free.
VAR
2 1
F 2

# One PSD constraint of this size:
#     | Two times two.
PSDCON
1
2

# One scalar constraint with an affine expression in this one conic domain:
#     | One is greater than or equal to zero.
CON
1 1
L+ 1

# Two coordinates in F^{obj}_j coefficients:
```

(continues on next page)

(continued from previous page)

```
#     | F^{obj}[0][0,0] = 1.0
#     | F^{obj}[0][1,1] = 1.0
OBJFCOORD
2
0 0 0 1.0
0 1 1 1.0

# Two coordinates in a^{obj}_j coefficients:
#     | a^{obj}[0] = 1.0
#     | a^{obj}[1] = 1.0
OBJACOORD
2
0 1.0
1 1.0

# One coordinate in b^{obj} coefficient:
#     | b^{obj} = 1.0
OBJBCOORD
1.0

# One coordinate in F_ij coefficients:
#     | F[0,0][1,0] = 1.0
FCOORD
1
0 0 1 0 1.0

# Two coordinates in a_ij coefficients:
#     | a[0,0] = -1.0
#     | a[0,1] = -1.0
ACOORD
2
0 0 -1.0
0 1 -1.0

# Four coordinates in H_ij coefficients:
#     | H[0,0][1,0] = 1.0
#     | H[0,0][1,1] = 3.0
#     | and more...
HCOORD
4
0 0 1 0 1.0
0 0 1 1 3.0
0 1 0 0 3.0
0 1 1 0 1.0

# Two coordinates in D_i coefficients:
#     | D[0][0,0] = -1.0
#     | D[0][1,1] = -1.0
DCOORD
2
0 0 0 -1.0
0 1 1 -1.0
```

Optimization Over a Sequence of Objectives

The linear optimization problem (16.14), is defined for a sequence of objectives such that hotstarting from one to the next might be advantages.

$$
\begin{array}{lll}
\text{maximize}_k & g_k^{obj} & \\
\text{subject to} & 50\,x_0 + 31 \quad \leq \quad 250\,, & \\
 & 3\,x_0 - 2x_1 \quad \geq \quad -4\,, & \\
 & x \in \mathbb{R}_+^2, &
\end{array} \tag{16.14}
$$

given,

1. $g_0^{obj} = x_0 + 0.64x_1$.
2. $g_1^{obj} = 1.11x_0 + 0.76x_1$.
3. $g_2^{obj} = 1.11x_0 + 0.85x_1$.

Its formulation in the CBF format is reported in Listing 16.5.

Listing 16.5: Problem (16.14) in CBF format.

```
# File written using this version of the Conic Benchmark Format:
#     | Version 1.
VER
1

# The sense of the objective is:
#     | Maximize.
OBJSENSE
MAX

# Two scalar variables in this one conic domain:
#     | Two are nonnegative.
VAR
2 1
L+ 2

# Two scalar constraints with affine expressions in these two conic domains:
#     | One is in the nonpositive domain.
#     | One is in the nonnegative domain.
CON
2 2
L- 1
L+ 1

# Two coordinates in a^{obj}_j coefficients:
#     | a^{obj}[0] = 1.0
#     | a^{obj}[1] = 0.64
OBJACOORD
2
0 1.0
1 0.64

# Four coordinates in a_ij coefficients:
#     | a[0,0] = 50.0
#     | a[1,0] = 3.0
#     | and more...
ACOORD
4
0 0 50.0
1 0 3.0
0 1 31.0
1 1 -2.0

# Two coordinates in b_i coefficients:
#     | b[0] = -250.0
#     | b[1] = 4.0
BCOORD
2
0 -250.0
1 4.0
```

(continues on next page)

(continued from previous page)

```
# New problem instance defined in terms of changes.
CHANGE

# Two coordinate changes in a^{obj}_j coefficients. Now it is:
#     | a^{obj}[0] = 1.11
#     | a^{obj}[1] = 0.76
OBJACOORD
2
0 1.11
1 0.76

# New problem instance defined in terms of changes.
CHANGE

# One coordinate change in a^{obj}_j coefficients. Now it is:
#     | a^{obj}[0] = 1.11
#     | a^{obj}[1] = 0.85
OBJACOORD
1
1 0.85
```

16.5 The PTF Format

The PTF format is a new human-readable, natural text format. Its features and structure are similar to the *OPF* format, with the difference that the PTF format **does** support semidefinite terms.

16.5.1 The overall format

The format is indentation based, where each section is started by a head line and followed by a section body with deeper indentation that the head line. For example:

```
Header line
   Body line 1
   Body line 1
   Body line 1
```

Section can also be nested:

```
Header line A
   Body line in A
   Header line A.1
      Body line in A.1
      Body line in A.1
   Body line in A
```

The indentation of blank lines is ignored, so a subsection can contain a blank line with no indentation. The character # defines a line comment and anything between the # character and the end of the line is ignored.

In a PTF file, the first section must be a `Task` section. The order of the remaining section is arbitrary, and sections may occur multiple times or not at all.

MOSEK will ignore any top-level section it does not recognize.

Names

In the description of the format we use following definitions for name strings:

```
NAME: PLAIN_NAME | QUOTED_NAME
PLAIN_NAME: [a-zA-Z_] [a-zA-Z0-9_-.!|]
QUOTED_NAME: "'" ( [^'\\\r\n] | "\\" ( [\\rn] | "x" [0-9a-fA-F] [0-9a-fA-F] ) )* "'"
```

Expressions

An expression is a sum of terms. A term is either a linear term (a coefficient and a variable name, where the coefficient can be left out if it is 1.0), or a matrix inner product.

An expression:

```
EXPR: EMPTY | [+-]? TERM ( [+-] TERM )*
TERM: LINEAR_TERM | MATRIX_TERM
```

A linear term

```
LINEAR_TERM: FLOAT? NAME
```

A matrix term

```
MATRIX_TERM: "<" FLOAT? NAME ( [+-] FLOAT? NAME)* ";" NAME ">"
```

Here the right-hand name is the name of a (semidefinite) matrix variable, and the left-hand side is a sum of symmetric matrixes. The actual matrixes are defined in a separate section.

Expressions can span multiple lines by giving subsequent lines a deeper indentation.

For example following two section are equivalent:

```
# Everything on one line:
x1 + x2 + x3 + x4

# Split into multiple lines:
x1
  + x2
  + x3
  + x4
```

16.5.2 `Task` section

The first section of the file must be a `Task`. The text in this section is not used and may contain comments, or meta-information from the writer or about the content.

Format:

```
Task NAME
   Anything goes here...
```

`NAME` is a the task name.

16.5.3 `Objective` section

The `Objective` section defines the objective name, sense and function. The format:

```
"Objective" NAME?
   ( "Minimize" | "Maximize" ) EXPR
```

For example:

```
Objective 'obj'
   Minimize x1 + 0.2 x2 + < M1 ; X1 >
```

16.5.4 `Constraints` section

The constraints section defines a series of constraints. A constraint defines a term $A \cdot x + b \in K$. For linear constraints `A` is just one row, while for conic constraints it can be multiple rows. If a constraint spans multiple rows these can either be written inline separated by semi-colons, or each expression in a separete sub-section.

Simple linear constraints:

```
"Constraints"
  NAME? "[" [-+] (FLOAT | "Inf") (";" [-+] (FLOAT | "Inf") )? "]" EXPR
```

If the brackets contain two values, they are used as upper and lower bounds. It they contain one value the constraint is an equality.

For example:

```
Constraints
  'c1' [0;10] x1 + x2 + x3
  [0] x1 + x2 + x3
```

Constraint blocks put the expression either in a subsection or inline. The cone type (domain) is written in the brackets, and **MOSEK** currently supports following types:

- `SOC(N)` Second order cone of dimension `N`
- `RSOC(N)` Rotated second order cone of dimension `N`
- `PSD(N)` Symmetric positive semidefinite cone of dimension `N`. This contains `N*(N+1)/2` elements.
- `PEXP` Primal exponential cone of dimension 3
- `DEXP` Dual exponential cone of dimension 3
- `PPOW(N,P)` Primal power cone of dimension `N` with parameter `P`
- `DPOW(N,P)` Dual power cone of dimension `N` with parameter `P`
- `ZERO(N)` The zero-cone of dimension `N`.

```
"Constraints"
  NAME? "[" DOMAIN "]" EXPR_LIST
```

For example:

```
Constraints
   'K1' [SOC(3)] x1 + x2 ; x2 + x3 ; x3 + x1
   'K2' [RSOC(3)]
      x1 + x2
      x2 + x3
      x3 + x1
```

16.5.5 `Variables` section

Any variable used in an expression must be defined in a variable section. The variable section defines each variable domain.

```
"Variables"
   NAME "[" [-+] (FLOAT | "Inf") (";" [-+] (FLOAT | "Inf") )? "]"
   NAME "[" DOMAIN "]" NAMES

   For example, a linear variable
```

```
Variables
   x1 [0;Inf]
```

As with constraints, members of a conic domain can be listed either inline or in a subsection:

```
Variables
   k1 [SOC(3)] x1 ; x2 ; x3
   k2 [RSOC(3)]
      x1
      x2
      x3
```

16.5.6 Integer section

This section contains a list of variables that are integral. For example:

```
Integer
   x1 x2 x3
```

16.5.7 SymmetricMatrixes section

This section defines the symmetric matrixes used for matrix coefficients in matrix inner product terms. The section lists named matrixes, each with a size and a number of non-zeros. Only non-zeros in the lower triangular part should be defined.

```
"SymmetricMatrixes"
   NAME "SYMMAT" "(" INT ")"  ( "(" INT "," INT "," FLOAT ")" )*
   ...
```

For example:

```
SymmetricMatrixes
   M1 SYMMAT(3) (0,0,1.0) (1,1,2.0) (2,1,0.5)
   M2 SYMMAT(3)
      (0,0,1.0)
      (1,1,2.0)
      (2,1,0.5)
```

16.5.8 Solutions section

Each subsection defines a solution. A solution defines for each constraint and for each variable exactly one primal value and either one (for conic domains) or two (for linear domains) dual values. The values follow the same logic as in the **MOSEK** C API. A primal and a dual solution status defines the meaning of the values primal and dual (solution, certificate, unknown, etc.)

The format is this:

```
"Solutions"
   "Solution" WHICHSOL
      "ProblemStatus" PROSTA PROSTA?
   "SolutionStatus" SOLSTA SOLSTA?
   "Objective" FLOAT FLOAT
   "Variables"
      # Linear variable status: level, slx, sux
      NAME "[" STATUS "]" FLOAT (FLOAT FLOAT)?
      # Conic variable status: level, snx
      NAME
         "[" STATUS "]" FLOAT FLOAT?
         ...
   "Constraints"
      # Linear variable status: level, slx, sux
      NAME "[" STATUS "]" FLOAT (FLOAT FLOAT)?
      # Conic variable status: level, snx
      NAME
         "[" STATUS "]" FLOAT FLOAT?
         ...
```

Following values for `WHICHSOL` are supported:

- `interior` Interior solution, the result of an interior-point solver.
- `basic` Basic solution, as produced by a simplex solver.
- `integer` Integer solution, the solution to a mixed-integer problem. This does not define a dual solution.

Following values for `PROSTA` are supported:

- `unknown` The problem status is unknown
- `feasible` The problem has been proven feasible
- `infeasible` The problem has been proven infeasible
- `illposed` The problem has been proved to be ill posed
- `infeasible_or_unbounded` The problem is infeasible or unbounded

Following values for `SOLSTA` are supported:

- `unknown` The solution status is unknown
- `feasible` The solution is feasible
- `optimal` The solution is optimal
- `infeas_cert` The solution is a certificate of infeasibility
- `illposed_cert` The solution is a certificate of illposedness

Following values for `STATUS` are supported:

- `unknown` The value is unknown
- `super_basic` The value is super basic
- `at_lower` The value is basic and at its lower bound
- `at_upper` The value is basic and at its upper bound
- `fixed` The value is basic fixed
- `infinite` The value is at infinity

16.6 The Task Format

The Task format is **MOSEK**'s native binary format. It contains a complete image of a **MOSEK** task, i.e.

- Problem data: Linear, conic, semidefinite and quadratic data
- Problem item names: Variable names, constraints names, cone names etc.
- Parameter settings
- Solutions

There are a few things to be aware of:

- Status of a solution read from a file will *always* be unknown.
- Parameter settings in a task file *always override* any parameters set on the command line or in a parameter file.

The format is based on the *TAR* (USTar) file format. This means that the individual pieces of data in a `.task` file can be examined by unpacking it as a *TAR* file. Please note that the inverse may not work: Creating a file using *TAR* will most probably not create a valid **MOSEK** Task file since the order of the entries is important.

16.7 The JSON Format

MOSEK provides the possibility to read/write problems in valid JSON format.

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.

The official JSON website http://www.json.org provides plenty of information along with the format definition.

MOSEK defines two JSON-like formats:

- `jtask`
- `jsol`

Despite being text-based human-readable formats, *jtask* and *jsol* files will include no indentation and no new-lines, in order to keep the files as compact as possible. We therefore strongly advise to use JSON viewer tools to inspect *jtask* and *jsol* files.

16.7.1 *jtask* format

It stores a problem instance. The *jtask* format contains the same information as a *task format*. Even though a *jtask* file is human-readable, we do not recommend users to create it by hand, but to rely on **MOSEK**.

16.7.2 *jsol* format

It stores a problem solution. The *jsol* format contains all solutions and information items.

16.7.3 A *jtask* example

In Listing 16.6 we present a file in the *jtask* format that corresponds to the sample problem from `lo1.lp`. The listing has been formatted for readability.

Listing 16.6: A formatted *jtask* file for the `lo1.lp` example.

```
{
    "$schema":"http://mosek.com/json/schema#",
    "Task/INFO":{
        "taskname":"lo1",
        "numvar":4,
        "numcon":3,
        "numcone":0,
        "numbarvar":0,
        "numanz":9,
        "numsymmat":0,
        "mosekver":[
            8,
            0,
            0,
            9
        ]
    },
    "Task/data":{
        "var":{
            "name":[
                "x1",
                "x2",
                "x3",
```

(continues on next page)

(continued from previous page)

```
            "x4"
        ],
        "bk":[
            "lo",
            "ra",
            "lo",
            "lo"
        ],
        "bl":[
            0.0,
            0.0,
            0.0,
            0.0
        ],
        "bu":[
            1e+30,
            1e+1,
            1e+30,
            1e+30
        ],
        "type":[
            "cont",
            "cont",
            "cont",
            "cont"
        ]
    },
    "con":{
        "name":[
            "c1",
            "c2",
            "c3"
        ],
        "bk":[
            "fx",
            "lo",
            "up"
        ],
        "bl":[
            3e+1,
            1.5e+1,
                -1e+30
        ],
        "bu":[
            3e+1,
            1e+30,
            2.5e+1
        ]
    },
    "objective":{
        "sense":"max",
        "name":"obj",
        "c":{
            "subj":[
                0,
                1,
                2,
                3
            ],
            "val":[
                3e+0,
```

(continues on next page)

(continued from previous page)


```
                    1e+0,
                    5e+0,
                    1e+0
                ]
            },
            "cfix":0.0
        },
        "A":{
            "subi":[
                0,
                0,
                0,
                1,
                1,
                1,
                1,
                2,
                2
            ],
            "subj":[
                0,
                1,
                2,
                0,
                1,
                2,
                3,
                1,
                3
            ],
            "val":[
                3e+0,
                1e+0,
                2e+0,
                2e+0,
                1e+0,
                3e+0,
                1e+0,
                2e+0,
                3e+0
            ]
        }
    },
    "Task/parameters":{
        "iparam":{
            "ANA_SOL_BASIS":"ON",
            "ANA_SOL_PRINT_VIOLATED":"OFF",
            "AUTO_SORT_A_BEFORE_OPT":"OFF",
            "AUTO_UPDATE_SOL_INFO":"OFF",
            "BASIS_SOLVE_USE_PLUS_ONE":"OFF",
            "BI_CLEAN_OPTIMIZER":"OPTIMIZER_FREE",
            "BI_IGNORE_MAX_ITER":"OFF",
            "BI_IGNORE_NUM_ERROR":"OFF",
            "BI_MAX_ITERATIONS":1000000,
            "CACHE_LICENSE":"ON",
            "CHECK_CONVEXITY":"CHECK_CONVEXITY_FULL",
            "COMPRESS_STATFILE":"ON",
            "CONCURRENT_NUM_OPTIMIZERS":2,
            "CONCURRENT_PRIORITY_DUAL_SIMPLEX":2,
            "CONCURRENT_PRIORITY_FREE_SIMPLEX":3,
            "CONCURRENT_PRIORITY_INTPNT":4,
```


(continues on next page)

(continued from previous page)

```
        "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,
```

(continues on next page)

(continued from previous page)

```
            "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,
```

(continues on next page)

(continued from previous page)

```
        "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,
```

(continues on next page)

(continued from previous page)

```
        "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":"",
```

(continues on next page)

(continued from previous page)

```
            "DATA_FILE_NAME":"examples/tools/data/lo1.mps",
            "DEBUG_FILE_NAME":"",
            "INT_SOL_FILE_NAME":"",
            "ITR_SOL_FILE_NAME":"",
            "MIO_DEBUG_STRING":"",
            "PARAM_COMMENT_SIGN":"%%",
            "PARAM_READ_FILE_NAME":"",
            "PARAM_WRITE_FILE_NAME":"",
            "READ_MPS_BOU_NAME":"",
            "READ_MPS_OBJ_NAME":"",
            "READ_MPS_RAN_NAME":"",
            "READ_MPS_RHS_NAME":"",
            "SENSITIVITY_FILE_NAME":"",
            "SENSITIVITY_RES_FILE_NAME":"",
            "SOL_FILTER_XC_LOW":"",
            "SOL_FILTER_XC_UPR":"",
            "SOL_FILTER_XX_LOW":"",
            "SOL_FILTER_XX_UPR":"",
            "STAT_FILE_NAME":"",
            "STAT_KEY":"",
            "STAT_NAME":"",
            "WRITE_LP_GEN_VAR_NAME":"XMSKGEN"
        }
    }
}
```

16.8 The Solution File Format

MOSEK provides several solution files depending on the problem type and the optimizer used:

- *basis solution file* (extension `.bas`) if the problem is optimized using the simplex optimizer or basis identification is performed,
- *interior solution file* (extension `.sol`) if a problem is optimized using the interior-point optimizer and no basis identification is required,
- *integer solution file* (extension `.int`) if the problem contains integer constrained variables.

All solution files have the format:

```
NAME                : <problem name>
PROBLEM STATUS      : <status of the problem>
SOLUTION STATUS     : <status of the solution>
OBJECTIVE NAME      : <name of the objective function>
PRIMAL OBJECTIVE    : <primal objective value corresponding to the solution>
DUAL OBJECTIVE      : <dual objective value corresponding to the solution>
CONSTRAINTS
INDEX  NAME     AT ACTIVITY    LOWER LIMIT   UPPER LIMIT   DUAL LOWER   DUAL UPPER
?      <name>   ?? <a value>   <a value>     <a value>     <a value>    <a value>
VARIABLES
INDEX  NAME     AT ACTIVITY    LOWER LIMIT   UPPER LIMIT   DUAL LOWER   DUAL UPPER    CONIC␣
↪DUAL
?      <name>   ?? <a value>   <a value>     <a value>     <a value>    <a value>     <a value>
```

In the example the fields `?` and `<>` will be filled with problem and solution specific information. As can be observed a solution report consists of three sections, i.e.

- `HEADER` In this section, first the name of the problem is listed and afterwards the problem and solution status are shown. Next the primal and dual objective values are displayed.

- `CONSTRAINTS` For each constraint i of the form

$$l_i^c \leq \sum_{j=1}^{n} a_{ij} x_j \leq u_i^c, \tag{16.15}$$

  the following information is listed:
  - `INDEX`: A sequential index assigned to the constraint by **MOSEK**
  - `NAME`: The name of the constraint assigned by the user.
  - `AT`: The status of the constraint. In Table 16.4 the possible values of the status keys and their interpretation are shown.

Table 16.4: Status keys.

Status key	Interpretation
UN	Unknown status
BS	Is basic
SB	Is superbasic
LL	Is at the lower limit (bound)
UL	Is at the upper limit (bound)
EQ	Lower limit is identical to upper limit
**	Is infeasible i.e. the lower limit is greater than the upper limit.

  - `ACTIVITY`: the quantity $\sum_{j=1}^{n} a_{ij} x_j^*$, where x^* is the value of the primal solution.
  - `LOWER LIMIT`: the quantity l_i^c (see (16.15).)
  - `UPPER LIMIT`: the quantity u_i^c (see (16.15).)
  - `DUAL LOWER`: the dual multiplier corresponding to the lower limit on the constraint.
  - `DUAL UPPER`: the dual multiplier corresponding to the upper limit on the constraint.
- `VARIABLES` The last section of the solution report lists information about the variables. This information has a similar interpretation as for the constraints. However, the column with the header `CONIC DUAL` is included for problems having one or more conic constraints. This column shows the dual variables corresponding to the conic constraints.

Example: `lo1.sol`

In Listing 16.7 we show the solution file for the `lo1.opf` problem.

Listing 16.7: An example of `.sol` file.

```
NAME                : 
PROBLEM STATUS      : PRIMAL_AND_DUAL_FEASIBLE
SOLUTION STATUS     : OPTIMAL
OBJECTIVE NAME      : obj
PRIMAL OBJECTIVE    : 8.33333333e+01
DUAL OBJECTIVE      : 8.33333332e+01

CONSTRAINTS
INDEX      NAME          AT ACTIVITY                 LOWER LIMIT        UPPER LIMIT        ␣
↪DUAL LOWER               DUAL UPPER
0          c1            EQ 3.00000000000000e+01     3.00000000e+01     3.00000000e+01     -0.
↪00000000000000e+00     -2.49999999741653e+00
1          c2            SB 5.33333333049187e+01     1.50000000e+01     NONE               2.
↪09159033069640e-10      -0.00000000000000e+00
2          c3            UL 2.49999999842049e+01     NONE               2.50000000e+01     -0.
↪00000000000000e+00     -3.33333332895108e-01

VARIABLES
INDEX      NAME          AT ACTIVITY                 LOWER LIMIT        UPPER LIMIT        ␣
↪DUAL LOWER               DUAL UPPER
0          x1            LL 1.67020427038537e-09     0.00000000e+00     NONE               -4.
↪49999999528054e+00     -0.00000000000000e+00
```

(continues on next page)

(continued from previous page)

```
1          x2              LL 2.93510446211883e-09     0.00000000e+00     1.00000000e+01     -2.
↪16666666494915e+00    6.20868657679896e-10
2          x3              SB 1.49999999899424e+01     0.00000000e+00     NONE               -8.
↪79123177245553e-10    -0.00000000000000e+00
3          x4              SB 8.33333332273115e+00     0.00000000e+00     NONE               -1.
↪69795978848200e-09    -0.00000000000000e+00
```

Chapter 17

List of examples

List of examples shipped in the distribution of Optimization Toolbox for MATLAB:

Table 17.1: List of distributed examples

File	Description
`advs.m`	Advanced simplex hot-start examples
`affco1.m`	A simple problem using affine conic constraints
`affco2.m`	A simple problem using affine conic constraints
`callback.m`	An example of data/progress callback
`callback_handler.m`	Log handler definition for `callback.m`
`ceo1.m`	A simple conic exponential problem
`cqo1.m`	A simple conic quadratic problem
`feasrepairex1.m`	A simple example of how to repair an infeasible problem
`gp1.m`	A simple geometric program (GP) in conic form
`lo1.m`	A simple linear problem using msklpopt
`lo2.m`	A simple linear problem using mosekopt
`lo3.m`	A simple linear problem using linprog
`mico1.m`	A simple mixed-integer conic problem
`milo1.m`	A simple mixed-integer linear problem
`mioinitsol.m`	A simple mixed-integer linear problem with an initial guess
`normex.m`	Demonstrates least squares and other norm minimization problems
`parameters.m`	Shows how to set optimizer parameters and read information items
`portfolio_1_basic.m`	Portfolio optimization - basic Markowitz model
`portfolio_2_frontier.m`	Portfolio optimization - efficient frontier
`portfolio_3_impact.m`	Portfolio optimization - market impact costs
`portfolio_4_transcost.m`	Portfolio optimization - transaction costs
`portfolio_5_card.m`	Portfolio optimization - cardinality constraints
`pow1.m`	A simple power cone problem
`qcqo1.m`	A simple quadratically constrained quadratic problem
`qo1.m`	A simple quadratic problem
`qo2.m`	A simple quadratic problem
`reoptimization.m`	Demonstrate how to modify and re-optimize a linear problem
`response.m`	Demonstrates proper response handling
`rlo1.m`	Robust linear optimization example, part 1

Continued on next page

Table 17.1 – continued from previous page

File	Description
rlo2.m	Robust linear optimization example, part 2
sdo1.m	A simple semidefinite optimization problem
sensitivity.m	Sensitivity analysis performed on a small linear problem
sensitivity2.m	Sensitivity analysis performed on a small linear problem
simple.m	A simple I/O example: read problem from a file, solve and write solutions
solutionquality.m	Demonstrates how to examine the quality of a solution

Additional examples can be found on the **MOSEK** website and in other **MOSEK** publications.

Chapter 18

Interface changes

The section shows interface-specific changes to the **MOSEK** Optimization Toolbox for MATLAB in version 9.0. See the release notes for general changes and new features of the **MOSEK** Optimization Suite.

18.1 Backwards compatibility

- **Parameters.** Users who set parameters to tune the performance and numerical properties of the solver (termination criteria, tolerances, solving primal or dual, presolve etc.) are recommended to reevaluate such tuning. It may be that other, or default, parameter settings will be more beneficial in the current version. The hints in Sec. 8 may be useful for some cases.
- Remove all `Near` solution statuses i.e. `MSK_SOL_STA_NEAR_OPTIMAL`, `MSK_SOL_STA_NEAR_PRIM_INFEAS_CER`, etc. See Sec. 13.3.3.
- All functions related to the general nonlinear optimizer and Scopt have been removed. See Sec. 15.8.

18.2 New API

Introduced a possibility to specify *affine conic constraints* i.e. conic constraints of the form $Fx + g \in K$ directly. See Sec. 6.7 and Sec. 12.5 for details.

18.3 Parameters

Added

- *MSK_IPAR_INTPNT_PURIFY*
- *MSK_IPAR_LOG_LOCAL_INFO*
- *MSK_IPAR_MIO_CONIC_OUTER_APPROXIMATION*
- *MSK_IPAR_MIO_FEASPUMP_LEVEL*
- *MSK_IPAR_MIO_MAX_NUM_ROOT_CUT_ROUNDS*
- *MSK_IPAR_MIO_PROPAGATE_OBJECTIVE_CONSTRAINT*
- *MSK_IPAR_MIO_SEED*
- *MSK_IPAR_OPF_WRITE_LINE_LENGTH*
- *MSK_IPAR_PRESOLVE_MAX_NUM_PASS*
- *MSK_IPAR_PTF_WRITE_TRANSFORM*

- *MSK_IPAR_SIM_SEED*
- *MSK_IPAR_WRITE_COMPRESSION*

Removed

- MSK_DPAR_DATA_TOL_AIJ
- MSK_DPAR_INTPNT_NL_MERIT_BAL
- MSK_DPAR_INTPNT_NL_TOL_DFEAS
- MSK_DPAR_INTPNT_NL_TOL_MU_RED
- MSK_DPAR_INTPNT_NL_TOL_NEAR_REL
- MSK_DPAR_INTPNT_NL_TOL_PFEAS
- MSK_DPAR_INTPNT_NL_TOL_REL_GAP
- MSK_DPAR_INTPNT_NL_TOL_REL_STEP
- MSK_DPAR_MIO_DISABLE_TERM_TIME
- MSK_DPAR_MIO_NEAR_TOL_ABS_GAP
- MSK_DPAR_MIO_NEAR_TOL_REL_GAP
- MSK_IPAR_MIO_CONSTRUCT_SOL
- MSK_IPAR_MIO_MT_USER_CB
- MSK_IPAR_OPF_MAX_TERMS_PER_LINE
- MSK_IPAR_READ_DATA_COMPRESSED
- MSK_IPAR_READ_DATA_FORMAT
- MSK_IPAR_WRITE_DATA_COMPRESSED
- MSK_IPAR_WRITE_DATA_FORMAT

18.4 Constants

Added

- *"MSK_COMPRESS_ZSTD"*
- *"MSK_CT_DEXP"*
- *"MSK_CT_DPOW"*
- *"MSK_CT_PEXP"*
- *"MSK_CT_PPOW"*
- *"MSK_CT_ZERO"*
- *"MSK_DATA_FORMAT_PTF"*
- *"MSK_IINF_MIO_NUMBIN"*
- *"MSK_IINF_MIO_NUMBINCONEVAR"*

- *"MSK_IINF_MIO_NUMCONE"*
- *"MSK_IINF_MIO_NUMCONEVAR"*
- *"MSK_IINF_MIO_NUMCONT"*
- *"MSK_IINF_MIO_NUMCONTCONEVAR"*
- *"MSK_IINF_MIO_NUMDEXPCONES"*
- *"MSK_IINF_MIO_NUMDPOWCONES"*
- *"MSK_IINF_MIO_NUMINTCONEVAR"*
- *"MSK_IINF_MIO_NUMPEXPCONES"*
- *"MSK_IINF_MIO_NUMPPOWCONES"*
- *"MSK_IINF_MIO_NUMQCONES"*
- *"MSK_IINF_MIO_NUMRQCONES"*
- *"MSK_IINF_MIO_PRESOLVED_NUMBINCONEVAR"*
- *"MSK_IINF_MIO_PRESOLVED_NUMCONE"*
- *"MSK_IINF_MIO_PRESOLVED_NUMCONEVAR"*
- *"MSK_IINF_MIO_PRESOLVED_NUMCONTCONEVAR"*
- *"MSK_IINF_MIO_PRESOLVED_NUMDEXPCONES"*
- *"MSK_IINF_MIO_PRESOLVED_NUMDPOWCONES"*
- *"MSK_IINF_MIO_PRESOLVED_NUMINTCONEVAR"*
- *"MSK_IINF_MIO_PRESOLVED_NUMPEXPCONES"*
- *"MSK_IINF_MIO_PRESOLVED_NUMPPOWCONES"*
- *"MSK_IINF_MIO_PRESOLVED_NUMQCONES"*
- *"MSK_IINF_MIO_PRESOLVED_NUMRQCONES"*
- *"MSK_IINF_PURIFY_DUAL_SUCCESS"*
- *"MSK_IINF_PURIFY_PRIMAL_SUCCESS"*
- *"MSK_LIINF_MIO_ANZ"*

Removed

- MSK_DATAFORMAT_XML
- MSK_DINFITEM_MIO_HEURISTIC_TIME
- MSK_DINFITEM_MIO_OPTIMIZER_TIME
- MSK_IINFITEM_MIO_CONSTRUCT_NUM_ROUNDINGS
- MSK_IINFITEM_MIO_INITIAL_SOLUTION
- MSK_IINFITEM_MIO_NEAR_ABSGAP_SATISFIED
- MSK_IINFITEM_MIO_NEAR_RELGAP_SATISFIED

- MSK_LIINFITEM_MIO_SIM_MAXITER_SETBACKS
- MSK_MIONODESELTYPE_HYBRID
- MSK_MIONODESELTYPE_WORST
- MSK_PROBLEMTYPE_GECO
- MSK_PROSTA_NEAR_DUAL_FEAS
- MSK_PROSTA_NEAR_PRIM_AND_DUAL_FEAS
- MSK_PROSTA_NEAR_PRIM_FEAS
- MSK_SENSITIVITYTYPE_OPTIMAL_PARTITION
- MSK_SOLSTA_NEAR_DUAL_FEAS
- MSK_SOLSTA_NEAR_DUAL_INFEAS_CER
- MSK_SOLSTA_NEAR_INTEGER_OPTIMAL
- MSK_SOLSTA_NEAR_OPTIMAL
- MSK_SOLSTA_NEAR_PRIM_AND_DUAL_FEAS
- MSK_SOLSTA_NEAR_PRIM_FEAS
- MSK_SOLSTA_NEAR_PRIM_INFEAS_CER

18.5 Response Codes

Added

- *"MSK_RES_ERR_APPENDING_TOO_BIG_CONE"*
- *"MSK_RES_ERR_CBF_DUPLICATE_POW_CONES"*
- *"MSK_RES_ERR_CBF_DUPLICATE_POW_STAR_CONES"*
- *"MSK_RES_ERR_CBF_INVALID_DIMENSION_OF_CONES"*
- *"MSK_RES_ERR_CBF_INVALID_EXP_DIMENSION"*
- *"MSK_RES_ERR_CBF_INVALID_NUMBER_OF_CONES"*
- *"MSK_RES_ERR_CBF_INVALID_POWER"*
- *"MSK_RES_ERR_CBF_INVALID_POWER_CONE_INDEX"*
- *"MSK_RES_ERR_CBF_INVALID_POWER_STAR_CONE_INDEX"*
- *"MSK_RES_ERR_CBF_POWER_CONE_IS_TOO_LONG"*
- *"MSK_RES_ERR_CBF_POWER_CONE_MISMATCH"*
- *"MSK_RES_ERR_CBF_POWER_STAR_CONE_MISMATCH"*
- *"MSK_RES_ERR_CBF_UNHANDLED_POWER_CONE_TYPE"*
- *"MSK_RES_ERR_CBF_UNHANDLED_POWER_STAR_CONE_TYPE"*
- *"MSK_RES_ERR_CONE_PARAMETER"*
- *"MSK_RES_ERR_FORMAT_STRING"*
- *"MSK_RES_ERR_INVALID_FILE_FORMAT_FOR_CFIX"*

- *"MSK_RES_ERR_INVALID_FILE_FORMAT_FOR_FREE_CONSTRAINTS"*
- *"MSK_RES_ERR_INVALID_FILE_FORMAT_FOR_NONLINEAR"*
- *"MSK_RES_ERR_INVALID_FILE_FORMAT_FOR_RANGED_CONSTRAINTS"*
- *"MSK_RES_ERR_NUM_ARGUMENTS"*
- *"MSK_RES_ERR_PTF_FORMAT"*
- *"MSK_RES_ERR_SHAPE_IS_TOO_LARGE"*
- *"MSK_RES_ERR_SLICE_SIZE"*
- *"MSK_RES_ERR_TOO_SMALL_A_TRUNCATION_VALUE"*
- *"MSK_RES_WRN_EXP_CONES_WITH_VARIABLES_FIXED_AT_ZERO"*
- *"MSK_RES_WRN_POW_CONES_WITH_ROOT_FIXED_AT_ZERO"*

Removed

- MSK_RES_ERR_CANNOT_CLONE_NL
- MSK_RES_ERR_CANNOT_HANDLE_NL
- MSK_RES_ERR_INVALID_ACCMODE
- MSK_RES_ERR_INVALID_FILE_FORMAT_FOR_GENERAL_NL
- MSK_RES_ERR_NONLINEAR_FUNCTIONS_NOT_ALLOWED
- MSK_RES_ERR_NR_ARGUMENTS
- MSK_RES_ERR_OPEN_DL
- MSK_RES_ERR_USER_FUNC_RET
- MSK_RES_ERR_USER_FUNC_RET_DATA
- MSK_RES_ERR_USER_NLO_EVAL
- MSK_RES_ERR_USER_NLO_EVAL_HESSUBI
- MSK_RES_ERR_USER_NLO_EVAL_HESSUBJ
- MSK_RES_ERR_USER_NLO_FUNC
- MSK_RES_TRM_MIO_NEAR_ABS_GAP
- MSK_RES_TRM_MIO_NEAR_REL_GAP
- MSK_RES_WRN_CONSTRUCT_INVALID_SOL_ITG
- MSK_RES_WRN_CONSTRUCT_NO_SOL_ITG
- MSK_RES_WRN_CONSTRUCT_SOLUTION_INFEAS
- MSK_RES_WRN_NO_NONLINEAR_FUNCTION_WRITE

Bibliography

[AA95] E. D. Andersen and K. D. Andersen. Presolving in linear programming. *Math. Programming*, 71(2):221–245, 1995.

[AGMX96] E. D. Andersen, J. Gondzio, Cs. Mészáros, and X. Xu. Implementation of interior point methods for large scale linear programming. In T. Terlaky, editor, *Interior-point methods of mathematical programming*, pages 189–252. Kluwer Academic Publishers, 1996.

[ART03] E. D. Andersen, C. Roos, and T. Terlaky. On implementing a primal-dual interior-point method for conic quadratic optimization. *Math. Programming*, February 2003.

[AY96] E. D. Andersen and Y. Ye. Combining interior-point and pivoting algorithms. *Management Sci.*, 42(12):1719–1731, December 1996.

[And09] Erling D. Andersen. The homogeneous and self-dual model and algorithm for linear optimization. Technical Report TR-1-2009, MOSEK ApS, 2009. URL: http://docs.mosek.com/whitepapers/homolo.pdf.

[And13] Erling D. Andersen. On formulating quadratic functions in optimization models. Technical Report TR-1-2013, MOSEK ApS, 2013. Last revised 23-feb-2016. URL: http://docs.mosek.com/whitepapers/qmodel.pdf.

[BTN00] A. Ben-Tal and A. Nemirovski. Robust solutions of linear programming problems contaminated with uncertain data. *Math. Programming*, 88(3):411–424, 2000.

[BKVH07] S. Boyd, S.J. Kim, L. Vandenberghe, and A. Hassibi. A Tutorial on Geometric Programming. *Optimization and Engineering*, 8(1):67–127, 2007. Available at http://www.stanford.edu/~boyd/gp_tutorial.html.

[Chv83] V. Chvátal. *Linear programming.* W.H. Freeman and Company, 1983.

[GK00] Richard C. Grinold and Ronald N. Kahn. *Active portfolio management.* McGraw-Hill, New York, 2 edition, 2000.

[Naz87] J. L. Nazareth. *Computer Solution of Linear Programs.* Oxford University Press, New York, 1987.

[RTV97] C. Roos, T. Terlaky, and J. -Ph. Vial. *Theory and algorithms for linear optimization: an interior point approach.* John Wiley and Sons, New York, 1997.

[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.

[BenTalN01] A. Ben-Tal and A. Nemirovski. *Lectures on Modern Convex Optimization: Analysis, Algorithms, and Engineering Applications.* MPS/SIAM Series on Optimization. SIAM, 2001.

Symbol Index

Enumerations

Functions

Parameters

Response codes

Structures

Index

U

V