MOSEK API for MATLAB

Release 11.1.1(BETA)

MOSEK ApS

24 November 2025

Contents

Chapter 1

Introduction

The **MOSEK** Optimization Suite 11.1.1(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 API for MATLAB?

The API for MATLAB provides access to the key functionalities of **MOSEK** from a MATLAB environment and allows to formulate optimization models with a convenient, intuitive syntax.

The API for MATLAB provides access to:

- Linear Optimization (LO)
- Conic Quadratic (Second-Order Cone) Optimization (CQO, SOCO)
- Power Cone Optimization
- Conic Exponential Optimization (CEO)
- Mixed-Integer Optimization (MIO) including Disjunctive Constraints (DJC)

Chapter 2

Contact Information

Phone	+45 7174 9373	Office
	+45 7174 5700	Sales
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
Linkedin	https://www.linkedin.com/company/mosek-aps
Youtube	https://www.youtube.com/channel/UCvIyectEVLP31NXeD5mIbEw

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

Chapter 3

License Agreement

3.1 MOSEK end-user license agreement

Before using the **MOSEK** software, please read the license agreement available in the distribution at `<MSKHOME>/mosek/11.1/mosek-eula.pdf` or on the **MOSEK** website https://mosek.com/products/license-agreement. By using **MOSEK** you agree to the terms of that license agreement.

3.2 Third party licenses

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

zlib

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

Listing 3.2: *fplib* license.

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

{fmt}

MOSEK uses the formatting library *{fmt}* developed by Victor Zverovich obtained form github/fmt and distributed under the MIT license. The license agreement fot *{fmt}* is shown in Listing 3.3.

Listing 3.3: *{fmt}* license.

```
Copyright (c) 2012 - present, Victor Zverovich

Permission is hereby granted, free of charge, to any person obtaining
a copy of this software and associated documentation files (the "Software"),
to deal in the Software without restriction, including without limitation
the rights to use, copy, modify, merge, publish, distribute, sublicense,
and/or sell copies of the Software, and to permit persons to whom the Software
is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included
in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR
A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER
IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

Zstandard

MOSEK uses the *Zstandard* library developed by Facebook obtained from github/zstd. The license agreement for *Zstandard* is shown in Listing 3.4.

Listing 3.4: *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.
```

OpenSSL

MOSEK uses the LibReSSL library, which is build on *OpenSSL*. OpenSSL is included under the OpenSSL license, Listing 3.5, and the LibReSSL additions are licensed under the ISC license, Listing 3.6.

Listing 3.5: *OpenSSL* license

```
====================================================================
Copyright (c) 1998-2011 The OpenSSL Project.  All rights reserved.

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

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

2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in
```

(continues on next page)

(continued from previous page)

```
   the documentation and/or other materials provided with the
   distribution.

3. All advertising materials mentioning features or use of this
   software must display the following acknowledgment:
   "This product includes software developed by the OpenSSL Project
   for use in the OpenSSL Toolkit. (http://www.openssl.org/)"

4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to
   endorse or promote products derived from this software without
   prior written permission. For written permission, please contact
   openssl-core@openssl.org.

5. Products derived from this software may not be called "OpenSSL"
   nor may "OpenSSL" appear in their names without prior written
   permission of the OpenSSL Project.

6. Redistributions of any form whatsoever must retain the following
   acknowledgment:
   "This product includes software developed by the OpenSSL Project
   for use in the OpenSSL Toolkit (http://www.openssl.org/)"

THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY
EXPRESSED 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 OpenSSL PROJECT OR
ITS 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.
====================================================================

This product includes cryptographic software written by Eric Young
(eay@cryptsoft.com).  This product includes software written by Tim
Hudson (tjh@cryptsoft.com).
```

Listing 3.6: *ISC* license

```
Copyright (C) 1994-2017 Free Software Foundation, Inc.
Copyright (c) 2014 Jeremie Courreges-Anglas <jca@openbsd.org>
Copyright (c) 2014-2015 Joel Sing <jsing@openbsd.org>
Copyright (c) 2014 Ted Unangst <tedu@openbsd.org>
Copyright (c) 2015-2016 Bob Beck <beck@openbsd.org>
Copyright (c) 2015 Marko Kreen <markokr@gmail.com>
Copyright (c) 2015 Reyk Floeter <reyk@openbsd.org>
Copyright (c) 2016 Tobias Pape <tobias@netshed.de>

Permission to use, copy, modify, and/or distribute this software for
any purpose with or without fee is hereby granted, provided that the
above copyright notice and this permission notice appear in all
copies.
```

(continues on next page)

(continued from previous page)

```
THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL
WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE
AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL
DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR
PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER
TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
PERFORMANCE OF THIS SOFTWARE.
```

mimalloc

MOSEK uses the *mimalloc* memory allocator library from github/mimalloc. The license agreement for *mimalloc* is shown in Listing 3.7.

Listing 3.7: *mimalloc* license.

```
MIT License

Copyright (c) 2019 Microsoft Corporation, Daan Leijen

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

BLASFEO

MOSEK uses the *BLASFEO* linear algebra library developed by Gianluca Frison, obtained from github/blasfeo. The license agreement for *BLASFEO* is shown in Listing 3.8.

Listing 3.8: *blasfeo* license.

```
BLASFEO -- BLAS For Embedded Optimization.
Copyright (C) 2019 by Gianluca Frison.
Developed at IMTEK (University of Freiburg) under the supervision of Moritz Diehl.
All rights reserved.

The 2-Clause BSD License

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

1. Redistributions of source code must retain the above copyright notice, this
```

(continues on next page)

(continued from previous page)

```
   list of conditions and the following disclaimer.
2. 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.

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 OWNER 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.
```

oneTBB

MOSEK uses the *oneTBB* parallelization library which is part of *oneAPI* developed by Intel, obtained from github/oneTBB, licensed under the Apache License 2.0. The license agreement for *oneTBB* can be found in https://github.com/oneapi-src/oneTBB/blob/master/LICENSE.txt .

Chapter 4

Installation

In this section we discuss how to install and setup the **MOSEK** API 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 API for MATLAB can be used with MATLAB version R2021a or newer on `linux64x86` and `win64x86` and R2023b or newer on `osxaarch64`.

Locating files in the MOSEK Optimization Suite

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

Table 4.1: Relevant files for the API for MATLAB.

Relative Path	Description	Label
`<MSKHOME>/mosek/11.1/tools/platform/<PLATFORM>/bin`	MATLAB files and libraries	`<MATLABAPIDIR>`
`<MSKHOME>/mosek/11.1/tools/examples/matlab`	Examples	`<EXDIR>`

where `<MSKHOME>` is the parent folder in which the **MOSEK** Optimization Suite has been installed and `<PLATFORM>` is one of `linux64x86`, `win64x86` or `osxaarch64`.

Setting up the paths

To use API for MATLAB the path to the MATLAB files must be added via the `addpath` command in MATLAB. Use the command

```
addpath <MSKHOME>/mosek/11.1/tools/platform/linux64x86/bin      (linux64x86)
addpath <MSKHOME>/mosek/11.1/tools/platform/osxaarch64/bin      (osxaarch64)
addpath <MSKHOME>\mosek\11.1\tools\platform\win64x86\bin        (win64x86)
```

4.1 Testing the installation

You can verify that API for MATLAB works by executing

```
mosekcheck
```

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

```
>> mosekcheck
Matlab version    : 9.10.0.1739362 (R2021a)
Architecture      : GLNXA64
MOSEK MeX path    : /home/user/somepath/mosek/11.0/tools/platform/linux64x86/bin/
↪mosekenv.mexa64
MOSEK version     : 11.0.0
PTS license       : Yes.
PTON license      : Yes.
MOSEK works OK    : You can use MOSEK in MATLAB.
```

In case of errors extra debug and license path information can be obtained with:

```
mosekcheck(debug = true);
```

4.2 Troubleshooting

Missing library files such as libmosek64.11.0.dylib or similar

If you are using macOS and get an error such as

```
Library not loaded: libmosek64.11.0.dylib
Referenced from:
/Users/.../.../.../mosekenv.mexmaca64
Reason: image not found.
```

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\11.0\tools\platform\win64x86\bin\mosekenv.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\11.1\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') ';<MSKHOME>\Mosek\11.0\tools\platform\win64x86\bin']);
```

Adjust the path to match your **MOSEK** location.

See also the Installation guide for details.

Incorrect use of '=' operator

An error such as

```
Error: File: mosekmodel.m Line: 491 Column: 49
Incorrect use of '=' operator. To assign a value to a variable, use '='. To compare␣
↪values for equality, use '=='.
```

will appear if your MATLAB version is too old. See above for the minimal supported MATLAB version.

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 `mosekenv`

If you get the MATLAB error message

```
Undefined function or variable 'mosekenv'
```

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

Security exception in MacOS 10.15+ (Catalina)

If an attempt to run **MOSEK** on Mac OS 10.15 (Catalina) and later produces security exceptions (`developer cannot be verified` and similar) then use `xattr` to remove the quarantine attribute from all **MOSEK** executables and binaries. This can be done in one go with

```
xattr -dr com.apple.quarantine mosek
```

where `mosek` is the folder which contains the full **MOSEK** installation or **MOSEK** binaries. See https://themosekblog.blogspot.com/2019/12/macos-1015-catalina-mosek-installation.html for more information. If that does not help, use the system settings to allow running arbitrary unverified applications.

Chapter 5

Design Overview

5.1 Modeling

The API for MATLAB consists of two components:

- The main **conic interface** for specifying linear, conic and mixed-integer optimization problems in conic format:

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Fx + g \in \mathcal{D} \end{array}$$

  specified by describing the matrix F, vectors c, g and a list of domains $\mathcal{D}$. The domains define linear and conic constraints on the corresponding affine expressions in $Fx + g$. See Sec. 7.1 for an introductory tutorial.

- The auxiliary **linear/simplex** interface for users who want to exploit features of the simplex algorithm, such as warm-start and basic solution. See Sec. 7.7 for an introductory tutorial.

The main characteristics of the API for MATLAB are:

- **Simplicity**: once the problem data is assembled in matrix form, it is straightforward to input it into the optimizer.
- **Extensibility**: the data can be input all at once or in chunks, corresponding for instance to consecutive constraint blocks in the model.
- **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.

API 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. 11 for the precise formulations of problems **MOSEK** solves.

5.2 "Hello World!" in MOSEK

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

Create a mosekmodel structure

Optimization problems using API for MATLAB are specified using a *mosekmodel* class that describes the numerical data of the problem.

Retrieving the solutions

When the problem is set up, the optimizer is invoked with the call to *mosekmodel.solve*, and subsequently the solution can be checked with *mosekmodel.hassolution* and retrieved with *mosekmodel.getsolution*.

We refer also to Sec. 6 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 simple optimization problem

$$\begin{array}{ll} \text{minimize} & x + 2y \\ \text{subject to} & x \geq 2.0, \;\; y \geq 3.0 \\ & 3x - y \leq 1 \end{array}$$

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

Listing 5.1: "Hello World!" in MOSEK

```
%%
%  Copyright: Copyright (c) MOSEK ApS, Denmark. All rights reserved.
%
%  File:      helloworld.m
%
%  The most basic example of how to get started with MOSEK.
%

function [xx,prosta,solsta] = helloworld()
    model = mosekmodel(...
                  name = "helloworld", ...
                  objsense = "min", ...
                  objective = [ 1, 2 ]', ...
                  numvar = 2, ...
                  F = [ 1 0 ; ...
                        0 1 ; ...
                        3 -1 ], ...
                  domain = [ mosekdomain("greater than", rhs = 2), ...    % x >= 2,  
↪1st row of F
                             mosekdomain("greater than", rhs = 3), ...    % y >= 3,  
↪2nd row of F
                             mosekdomain("less than", rhs = 1) ]);        % 3x-y <= 1,
↪3rd row of F
    model.solve();

    if model.hassolution("interior")
        [xx,prosta,solsta] = model.getsolution("any", "x");
        xx
    end

end
```

Chapter 6

Solver Interaction Tutorials

In this section we cover the interaction with the solver.

6.1 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.

6.1.1 Solver termination

If an error occurs during optimization then an exception will be raised. More about errors and exceptions in Sec. 6.2.

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. 6.3.

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

6.1.2 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 6.1: Types of solutions available from **MOSEK**

	Simplex optimizer	Interior-point optimizer	Mixed-integer optimizer
Linear problem	`basic`	`interior`	
Nonlinear continuous problem		`interior`	
Problem with integer variables			`integer`

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.

6.1.3 Problem and solution status

Assuming that the optimization terminated without errors, the next important step is to check the problem and solution status and availability of solutions. 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 *"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** (*"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 outputs `prosta` and `solsta` of *moseкmodel.hassolution*.

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

Outcome	Problem status	Solution status
Optimal	*"PRIM_AND_DUAL_FEAS"*	*"OPTIMAL"*
Primal infeasible	*"PRIM_INFEAS"*	*"PRIM_INFEAS_CER"*
Dual infeasible (unbounded)	*"DUAL_INFEAS"*	*"DUAL_INFEAS_CER"*
Uncertain (stall, numerical issues, etc.)	*"UNKNOWN"*	*"UNKNOWN"*

Table 6.3: Integer problems (solution status for integer solution, others undefined)

Outcome	Problem status	Solution status
Integer optimal	*"PRIM_FEAS"*	*"INTEGER_OPTIMAL"*
Infeasible	*"PRIM_INFEAS"*	*"UNKNOWN"*
Integer feasible point	*"PRIM_FEAS"*	*"PRIM_FEAS"*
No conclusion	*"UNKNOWN"*	*"UNKNOWN"*

6.1.4 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:

- *mosekmodel.getsolution* — the primal or dual solution values for arguments x and y, respectively.

6.1.5 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 6.1: Sample framework for checking optimization result.

```
% Solve the model
try
    model.solve();
catch ME
    warning("An error during optimization; handle it here.");
    rethrow(ME);
end

% We check if the interior-point solution exists and what status it has
[exists, prosta, solsta] = model.hassolution("interior");

if exists
    disp("Solved the problem with statuses:");
    disp(prosta);
    disp(solsta);

    switch solsta
        case "OPTIMAL"
            disp("Optimal solution found:");
            x = model.getsolution("interior");
            disp(x);
        case "PRIM_INFEAS_CER"
            disp("The problem is primal infeasible.");
        case "DUAL_INFEAS_CER"
            disp("The problem is dual infeasible.");
        case "UNKNOWN"
            disp("Solution status UNKNOWN. This could indicate numerical issues");
        default
            disp("Another solution status:")
            disk(solsta)
    end
else
    warning("Solution does not exists");
end
```

6.2 Errors and exceptions

Exceptions and response codes

The functions of API for MATLAB will throw a MATLAB exception in case of errors. There are two types of exception messages:

- Simple calls to functions setting up the model will throw an exception with a single string describing the error, for example missing or incorrect data.
- Calls to functions such as *mosekmodel.solve* which invoke the **MOSEK** library will throw exceptions with a standardized string containing a **MOSEK** response code and an explanation (see below). The response code can be cross-referenced against the list of **MOSEK** response codes in Sec. 13.6.

For this reason it is a good idea to call **MOSEK** functions in a try-catch block. The one case where it is *extremely important* to check for an exception is when calling *mosekmodel.solve*. For more information see Sec. 6.1.

As an example, consider:

```
model = mosekmodel(objective = [NaN], numvar = 1);
try
  model.solve();
catch ME
  fprintf("An error during solve(). Message:\n%s\n", ME.message);
end
```

This will produce as output:

```
An error during solve(). Message:
MSK_RES_ERR_NAN_IN_C(1470):The objective vector c contains an invalid value for␣
↪variable '' (0).
```

In many cases (especially related to licensing) a much more verbose error message will be printed to the log.

Parsing the exception message.

An exception message produced by **MOSEK** during a call to *mosekmodel.solve* has the format

```
ERROR_NAME(ERROR_CODE):MESSAGE
```

where the ERROR_NAME is a string and ERROR_CODE is an integer, both of which can be cross-referenced against Sec. 13.6. The integer value is *not* guaranteed to remain constant between **MOSEK** versions, so it is recommended to test for equality of strings in ERROR_NAME.

The message can be parsed into individual components as follows:

```
model = mosekmodel(objective = [NaN], numvar = 1);
try
  model.solve();
catch ME
  expr = "([A-Z_]*)\(([0-9]*)\):(.*)";
  [tokens, matches] = regexp(ME.message, expr, "tokens", "match");
  fprintf("Error name: %s\n",   tokens{1}{1});
  fprintf("Code:       %.0f\n", str2double(tokens{1}{2}));
  fprintf("Message:    %s\n",   tokens{1}{3});
end
```

which leads to an output such as

```
Error name: MSK_RES_ERR_NAN_IN_C
Code:       1470
Message:    The objective vector c contains an invalid value for variable '' (0).
```

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. 6.3). A typical warning is, for example:

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

Error and solution status handling example

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

Listing 6.2: Sample framework for checking optimization result.

```
    % Solve the model
    try
        model.solve();
    catch ME
        warning("An error during optimization; handle it here.");
        rethrow(ME);
    end

    % We check if the interior-point solution exists and what status it has
    [exists, prosta, solsta] = model.hassolution("interior");

    if exists
        disp("Solved the problem with statuses:");
        disp(prosta);
        disp(solsta);

        switch solsta
            case "OPTIMAL"
                disp("Optimal solution found:");
                x = model.getsolution("interior");
                disp(x);
            case "PRIM_INFEAS_CER"
                disp("The problem is primal infeasible.");
            case "DUAL_INFEAS_CER"
                disp("The problem is dual infeasible.");
            case "UNKNOWN"
                disp("Solution status UNKNOWN. This could indicate numerical issues");
            default
                disp("Another solution status:")
                disk(solsta)
        end
    else
        warning("Solution does not exists");
    end
```

6.3 Input/Output

6.3.1 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 option `quiet`, for example:

```
model.solve(quiet = true);
```

6.3.2 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*.

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.

6.3.3 Saving a problem to a file

An optimization problem can be dumped to a file using the option `write_to_file` in *mosekmodel.solve* or using the dedicated method *mosekmodel.write*. The file format will be determined from the filename's extension. Supported formats are listed in Sec. 14 together with a table of problem types supported by each.

For instance the problem can be written to a human-readable PTF file (see Sec. 14.5) with

```
% Before solving:
model.solve(write_to_file = "dump.ptf");
% or without solving:
model.write("dump.ptf");
```

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

```
% Before solving:
model.solve(write_to_file = "dump.task.gz");
% or without solving:
model.write("dump.task.gz");
```

Some remarks:

- The problem is written to the file as it is represented in the underlying *optimizer task*.
- 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.

6.3.4 Reading a problem from a file

It is not possible to read a file saved with `write_to_file` back into a API for MATLAB data structure. However, such problem files can be solved with the command-line tool or read by the low-level Optimizer API if necessary. See the documentation of those interfaces for details.

6.4 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 the parameter value should always be passed as a string. Parameters are passed through the argument `param` in *mosekmodel.solve* as an array of name-value pairs `[name1, value1, name2, value2, ...]`.

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 some parameters before solving a problem.

Listing 6.3: Parameter setting example.

```
% Solve with a list of parameters
model.solve(param = ["MSK_IPAR_LOG", "1", ...                          % Set log␣
↪level (integer parameter)
                     "MSK_IPAR_CACHE_LICENSE", "MSK_OFF", ...          % Do not␣
↪keep the license (integer parameter)
                     "MSK_DPAR_INTPNT_CO_TOL_REL_GAP", "1.0e-7" ]);    % Set␣
↪relative gap tolerance (double parameter)
```

6.5 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*

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 returned in the `info` field of the model object.

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 6.4: Information items example.

```
fprintf("Optimizer time %.3f\n", model.info.MSK_DINF_OPTIMIZER_TIME);
fprintf("#iterations    %d\n",   model.info.MSK_IINF_INTPNT_ITER);
```

6.6 MOSEK OptServer

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

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

The `URL` of the remote server required in all client-side calls should be a string of the form `http://host:port` or `https://host:port`.

6.6.1 Synchronous Remote Optimization

In synchronous mode the client sends an optimization problem to the server and blocks, waiting for the optimization to end. Once the result has been received, the program can continue. This is the simplest mode all it takes is to provide the address of the server before starting optimization. The rest of the code remains untouched.

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

Source code example

Listing 6.5: Using the OptServer in synchronous mode.

```
function opt_server_sync(url, cert)

% Here we can set up a model
model = mosekmodel(...
              objsense = "min", ...
              objective = [ 1, 2 ]', ...
              numvar = 2, ...
              F = [ 1 0 ; ...
                    0 1 ; ...
                    3 -1 ], ...
              domain = [ mosekdomain("greater than", rhs = 2), ...
                         mosekdomain("greater than", rhs = 3), ...
                         mosekdomain("less than", rhs = 1) ]);


% Set up the certificate path, if using TLS, otherwise ignore
if exist('cert','var')
        param = ["MSK_SPAR_REMOTE_TLS_CERT_PATH", cert ];
else
        param = [];
end

% Optimize using the remote server
model.solve(param = param, optserver = url);

% Use the optimal solution
if model.hassolution("interior")
        xx = model.getsolution("interior", "x");
        disp(xx);
end
```

Chapter 7

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.

- **The conic interface tutorial**
  - Sec. 7.1. Linear and conic optimization tutorial using the conic interface. **Essential first reading for all users**. Shows all the steps of setting up a conic or linear optimization problem, solving it and retrieving the solutions.

  Further basic examples covering a variety of cones:
  - Sec. 7.2. Exponential cone, power cone and cone combinations in various settings.
  - Sec. 7.3. A basic tutorial of geometric programming (GP).
- **Mixed-integer optimization tutorials (MIO)**
  - Sec. 7.4. Shows how to declare integer variables for linear and conic problems and how to set an initial solution.
  - Sec. 7.5. Demonstrates how to create a problem with disjunctive constraints (DJC).
- **Infeasibility certificates**
  - Sec. 7.6. Shows how to retrieve and analyze a primal infeasibility certificate for continuous problems.
- **The linear/simplex interface tutorial**
  - Sec. 7.7. Tutorial for the linear optimization toolbox, intended mainly to solve linear problems when the simplex algorithm is used, basic solution is required or warm-start is exploited.
  - Sec. 7.8. Tutorial for warm-starting the simplex optimizer with the linear optimization toolbox.

7.1 The conic interface tutorial

In this tutorial we demonstrate how to set up and solve a linear or conic problem with `mosekmodel`, the main interface of the API for MATLAB.

A problem solved with `mosekmodel` has the form

$$\begin{array}{lll} \text{minimize/maximize} & c^T x + c^f & \\ \text{subject to} & Fx + g & \in \quad \mathcal{D}, \end{array} \tag{7.1}$$

where

- 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
- $F \in \mathbb{R}^{k \times n}$ is the affine conic constraint matrix.,
- $g \in \mathbb{R}^k$ is the affine conic constraint constant term vector.,
- $\mathcal{D}$ is a domain of dimension k, constructed from the Sec. 13.8.

Let us indicate a few syntactic features of the API for MATLAB, which allow specifying such poblems with greater flexibility:

- **Compact input with one call.** The problem data has only three major elements, `F,g,c`, so if these matrices are constructed in advance, the problem can be set up with a single call to *moseckmodel*.
- **Setting up constraint blocks separately.** Alternatively, as it is in most practical applications, the constraint section may consist of blocks, each corresponding to some logical part of the problem specification. We refer to them as *affine conic constraints* (ACCs). In this case it may be more natural to enter individual ACCs

$$F_i x + g_i \in \mathcal{D}_i$$

  separately, using the method *mosekmodel.appendcons*. The API for MATLAB will then internally combine these blocks into the F, g data by vertical stacking and let $\mathcal{D} = \mathcal{D}_1 \times \cdots \times \mathcal{D}_p$, where p is the number of ACCs.
- **Linear problems.** The general formulation in (7.1) includes standard linear problems by using linear domains. For example $Ax+b \leq 0$ is expressed as $Ax+b \in \mathbb{R}_{\leq 0}$ and $Ax+b = 0$ as $Ax+b \in \{0\}$, see Sec. 13.8.
- **Shifted domains.** To simplify some constructions a domain may be shifted by a vector b, so that effectively one can write

$$Fx + g \in \mathcal{D} + b.$$

  This is a syntactic feature as one could achieve the same effect replacing g with $g - b$, but it is sometimes more natural; especially for linear problems it may be more intuitive to write $Ax \geq b$ as $Ax \in \mathbb{R}_{\geq 0} + b$ rather than $Ax - b \in \mathbb{R}_{\geq 0}$. For obvious reasons this argument is called `rhs` (right-hand side) when creating a domain with *mosekdomain*.

After this introduction we proceed to demonstrate examples of linear and conic problems, set up either at once or in chunks. We then finish the tutorial demonstrating how to invoke the optimizer and retrieve the solutions.

7.1.1 Linear example (LO1)

We set up a linear problem in one go. Consider the problem

$$\begin{array}{lccccccccl} \text{maximize} & 3x_0 & + & 1x_1 & + & 5x_2 & + & 1x_3 & & \\ \text{subject to} & 3x_0 & + & 1x_1 & + & 2x_2 & & & = & 30, \\ & 2x_0 & + & 1x_1 & + & 3x_2 & + & 1x_3 & \geq & 15, \\ & & & 2x_1 & & & + & 3x_3 & \leq & 25, \\ & x_0, & & x_1, & & x_2, & & x_3 & \geq & 0, \\ & & & x_1 & & & & & \leq & 10. \end{array} \tag{7.2}$$

We create a model by providing all data at once:

- the number of variables is 4 and is set as `numvar`,
- the objective vector is $c = [3, 1, 5, 1]$ and is passed as `objective`,
- the objective sense is maximization, and is passed as `objsense`,
- the problem name `name` (optional) is `lo1`.

Listing 7.1: Setting up a linear model in a single call.

```
model = mosekmodel(...
            name = "lo1", ...
            objsense = "maximize", ...
            objective = [ 3 1 5 1 ]', ...
            numvar = 4, ...
            F = [ 3 1 2 0 ; ...
                  2 1 3 1 ; ...
                  0 2 0 3 ; ...
                  1 0 0 0 ; ...
                  0 1 0 0 ; ...
                  0 0 1 0 ; ...
                  0 0 0 1 ; ...
                  0 1 0 0 ;], ...
            domain = [ mosekdomain("equal",        rhs=30), ...
                       mosekdomain("greater than", rhs=15), ...
                       mosekdomain("less than",    rhs=25), ...
                       mosekdomain("nonnegative",  n=4), ...
                       mosekdomain("less than",    rhs=10) ]);
```

The remaining data is the matrix F, each row of F corresponding to one linear bound in the problem. For each of those bounds there is a corresponding domain in the list of domains `domain` which indicates the type of (in)equality (lower-bounded, upper-bounded, equals). We exploit the `rhs` vector mentioned in the introduction to pass the bounds inside domains, rather than having to add a `g` vector, although both options would be just as good. Note that the bounds

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

are covered by one domain $\mathbb{R}^4_{\geq 0}$, that is of dimension `n=4` to indicate the number of rows of F it covers. All other bounds are (by default) 1-dimensional, i.e. correspond to a single row of `F`. The total dimension of all domains $(1 + 1 + 1 + 4 + 1 = 8)$ equals the number of rows in F.

In large, practical applications the matrix F can, and should, be specified as a sparse matrix.

See the API reference for *mosekmodel* for a specification of all possible arguments.

7.1.2 Linear example with multiple calls (LO2)

We can set up the model of (7.2) adding linear constraints one by one with multiple calls to *mosekmodel.appendcons*. In the example below we first initialize the model object with the name, objective and number of variables, and then we add the linear constraints separately

Listing 7.2: Setting up a linear model with multiple calls.

```
model = mosekmodel(name = "lo2", ...
                   objsense = "maximize", ...
                   objective = [ 3 1 5 1 ]', ...
                   numvar = 4);

model.appendcons(name="con-eq30", F = [ 3 1 2 0 ], domain = mosekdomain("equal", ␣
↪     rhs=30));
model.appendcons(name="con-gt14", F = [ 2 1 3 1 ], domain = mosekdomain("greater␣
↪than", rhs=15));
model.appendcons(name="con-lt25", F = [ 0 2 0 3 ], domain = mosekdomain("less than
↪",    rhs=25));
model.appendcons(name="con-nneg", F = speye(4),    domain = mosekdomain(
↪"nonnegative", n=4));
model.appendcons(name="con-lt10", F = [ 0 1 0 0 ], domain = mosekdomain("less than
↪", rhs="10"));
```

Each call to *mosekmodel.appendcons* contains the constraint's name (optional) and it's F matrix and domain as before. In each call to *mosekmodel.appendcons* the number of rows in F equals the dimension of the domain.

7.1.3 Conic quadratic example (ACC1)

We now go through an example with non-linear conic constraints, in this case quadratic. All other cones would be added in a similar way. Consider the problem

$$\begin{array}{ll} \text{maximize} & c^T x \\ \text{subject to} & \sum_i x_i = 1, \\ & \gamma \geq \|Gx + h\|_2, \end{array} \tag{7.3}$$

where $x \in \mathbb{R}^n$ is the optimization variable and $G \in \mathbb{R}^{k \times n}$, $h \in \mathbb{R}^k$, $c \in \mathbb{R}^n$ and $\gamma \in \mathbb{R}$.

The norm constraint has a conic representation:

$$\begin{array}{ll} \text{maximize} & c^T x \\ \text{subject to} & \sum_i x_i = 1, \\ & (\gamma, Gx + h) \in \mathcal{Q}^{k+1}, \end{array} \tag{7.4}$$

and we can write it explicitly in matrix format $Fx + g \in \mathcal{D}$ as follows:

$$\begin{bmatrix} 0 \\ G \end{bmatrix} x + \begin{bmatrix} \gamma \\ h \end{bmatrix} \in \mathcal{Q}^{k+1}.$$

We formulate the problem by including the linear constraint and the conic constraint in two separate calls.

Listing 7.3: Setting up a conic quadratic model.

```
% Initialize the model
model = mosekmodel(name = "acc1",...
                   numvar = n);

% Set objective vector
model.objective("maximize", c);

% The constraint sum(x) = 1
model.appendcons(F = ones(1,n), domain = mosekdomain("equal", rhs=1));

% The conic quadratic constraint Fx+g \in \Quad with k+1 rows
model.appendcons(F = sparse([zeros(1,n); G]), ...
                 g = [gamma; h], ...
                 domain = mosekdomain("quadratic cone", dim=k+1));
```

7.1.4 Solving and retrieving the solution

We wrap up with a short demonstration of what to do after the model has been defined, that is how to solve the model and retrieve solutions.

Listing 7.4: Solving and retrieving the solution.

```
% Solve the problem
model.solve();

% Check if solution is available
[hassol, prosta, solsta] = model.hassolution("interior");

if hassol && solsta == "OPTIMAL"
```

(continues on next page)

(continued from previous page)

```
        % Get primal solution
        xx = model.getsolution("interior", "x");

        % Get dual solution
        y = model.getsolution("interior", "y");

        disp("Primal solution");
        disp(xx);
    end
```

We note that

- The optimizer is invoked with *mosekmodel.solve*.
- We check if the interior point solution is available with *mosekmodel.hassolution*, which returns also problem and solution status.
- The primal and dual solution are obtained by requesting the "x" or "y" component in *mosekmodel.getsolution*.
- See Sec. 6.1 for more details about retrieving solutions and handling more solution statuses, and Sec. 6.2 for information about error handling, which we omitted for readability.

7.2 A gallery of conic examples

In this chapter we demonstrate various simple examples of conic problems including various cone types (domains). We assume full familiarity with the basic tutorial of Sec. 7.1.

7.2.1 Example CQO1 (quadratic cones)

We solve the conic quadratic problem:

$$\begin{array}{ll} \text{minimize} & y_1 + y_2 + y_3 \\ \text{subject to} & x_1 + x_2 + 2.0x_3 = 1.0, \\ & x_1, x_2, x_3 \geq 0.0, \\ & (y_1, x_1, x_2) \in \mathcal{Q}^3, \\ & (y_2, y_3, x_3) \in \mathcal{Q}_r^3. \end{array} \tag{7.5}$$

The variable has length 6 and is ordered as x followed by y from (7.5). We add the constraints in the order in which they appear above.

Listing 7.5: Source code solving problem (7.5).

```
    model = mosekmodel(...
        name = "cqo1", ...
        numvar = 6);

    % Variable is [x, y]
    model.objective("minimize", [0 0 0 1 1 1]');

    % Linear constraint
    model.appendcons(F = [1 1 2 0 0 0], domain = mosekdomain("eq", rhs=1.0));

    % Bounds on x
    model.appendcons(F = sparse([eye(3) zeros(3)]), domain = mosekdomain("rplus",␣
↪dim=3));

    % Quadratic cone
```

(continues on next page)

(continued from previous page)

```
    model.appendcons(F = sparse([1,2,3],[4,1,2],[1,1,1]), domain = mosekdomain("qcone
↪",dim=3));

    % Rotated quadratic cone
    model.appendcons(F = sparse([1,2,3],[5,6,3],[1,1,1]), domain = mosekdomain("rqcone
↪",dim=3));

    model.solve();
    if model.hassolution("interior")
        [xx,prosta,solsta] = model.getsolution("interior","x");
        fprintf("Solution x1,x2,x3: [ %s ]\n", sprintf("%.2g ", xx(1:3)));
        fprintf("Solution y1,y2,y3: [ %s ]\n", sprintf("%.2g ", xx(4:6)));
    end
```

7.2.2 Example AFFCO2 (power cones and auxiliary variables)

Consider the following simple optimization problem:

$$\begin{array}{lrcl} \text{maximize} & x_1^{1/3} + (x_1 + x_2 + 0.1)^{1/4} & & \\ \text{subject to} & (x_1 - 0.5)^2 + (x_2 - 0.6)^2 & \leq & 1, \\ & x_1 - x_2 & \leq & 1. \end{array} \tag{7.6}$$

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

$$\begin{array}{lrcl} \text{maximize} & t_1 + t_2 & & \\ \text{subject to} & (1, x_1 - 0.5, x_2 - 0.6) & \in & \mathcal{Q}^3, \\ & (x_1, 1, t_1) & \in & \mathcal{P}_3^{(1/3,2/3)}, \\ & (x_1 + x_2 + 0.1, 1, t_2) & \in & \mathcal{P}_3^{(1/4,3/4)}, \\ & x_1 - x_2 & \leq & 1. \end{array} \tag{7.7}$$

We arrange the variables as `x` followed by `t`.

For the sake of demonstration:

- we add constraints in three blocks: first the quadratic constraint, then both power cone constraints together, and finally the linear constraint,
- we add the quadratic constraint in the constructor *mosekmodel* and the other ones later using *mosekmodel.appendcons*,

while of course any other combination is also possible.

For example, the constraint $(1, x_1 - 0.5, x_2 - 0.6) \in \mathcal{Q}^3$ is written in matrix form as

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

The joint power cone constraints have the following representation:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ t_1 \\ t_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0.1 \\ 1 \\ 0 \end{bmatrix} \in \mathcal{P}_3^{(1/3,2/3)} \times \mathcal{P}_3^{(1/4,3/4)}.$$

This is altogether implemented as follows:

Listing 7.6: Script implementing conic version of problem (7.6).

```
% Variables [x1; x2; t1; t2]
model = mosekmodel(name = "affco1", numvar = 4, ...
                   objsense = "maximize", objective = [0, 0, 1, 1], ...
                   F = sparse([zeros(1,4); speye(2) zeros(2,2)]), ...    % The␣
↪quadratic cone constraint
                   g = [1 -0.5 -0.6]', ...
                   domain = mosekdomain("qcone", dim = 3));

% The power cones added as one block:
model.appendcons(name="pow", ...
                 F = sparse([1,3,4,4,6], [1,3,1,2,4], ones(1,5)), ...
                 g = [0 1 0 0.1 1 0]', ...
                 domain = [mosekdomain("pow", dim = 3, alpha = [1 2]'), ...     %␣
↪Exponents [ 1/3, 2/3 ]
                         mosekdomain("pow", dim = 3, alpha = 0.25) ]);         %␣
↪Exponents [ 0.25, 0.75 ]

% Linear inequality x_1 - x_2 <= 1
model.appendcons(F = [1 -1 0 0], domain = mosekdomain("less than", rhs = 1));
```

7.2.3 Example AFFCO2 (many exponential cones)

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:

$$\begin{array}{lrcl} \text{minimize} & & & t \\ \text{subject to} & (d, z_1 y_1, \ldots, z_n y_n) & \in & \mathcal{Q}^{n+1}, \\ & (y_i, 1, a_i t) & \in & K_{\exp},\ i = 1, \ldots, n, \end{array} \tag{7.8}$$

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 the affine conic representation is

$$\begin{bmatrix} 0 & 0_n^T \\ 0_n & \mathbf{Diag}(z_1, \ldots, z_n) \end{bmatrix} \begin{bmatrix} t \\ y \end{bmatrix} + \begin{bmatrix} d \\ 0_n \end{bmatrix} \in \mathcal{Q}^{n+1}.$$

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. We assemble all those exponential cone desctiptions in one call to *mosekmodel.appendcons*, where the argument `n` denotes the number of cones.

Listing 7.7: Script implementing problem (7.8).

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

   % Variables [t, y1, ..., yn]
   model = mosekmodel(...
       name = "affco2", ...
       numvar = n + 1, ...
       objective = [1 zeros(1,n)], ...
       objsense = "minimize");

   model.varname([1], ["t"]);

   % Quadratic cone
   model.appendcons(F = diag([0; z]), ...
                    g = [d; zeros(n,1)], ...
                    domain = mosekdomain("qcone", dim = n + 1));

   % All exponential cones (their number is n)
   FExp = sparse([1:3:3*n    3:3:3*n], ...
                 [2:n+1      ones(1,n)], ...
                 [ones(1,n)  a']);
   gExp = repmat([0; 1; 0], n, 1);
   model.appendcons(F = FExp, g = gExp, ...
                    domain = mosekdomain("exp", dim = 3, n = n));

   % Solve and get solution
   model.solve();

   if model.hassolution("interior")
      [x, prosta, solsta] = model.getsolution("interior", "x");
      t = x(1);
   end
```

7.3 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

$$\begin{array}{lll} \text{minimize} & f_0(x) & \\ \text{subject to} & f_i(x) \leq 1, & i = 1, \ldots, m, \\ & x_j > 0, & j = 1, \ldots, n, \end{array} \tag{7.9}$$

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

$$\log(\sum_k \exp(a_k^T y + b_k)) \leq 0 \tag{7.10}$$

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:

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

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.

7.3.1 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:

$$\begin{array}{lrl} \text{maximize} & hwd & \\ \text{subject to} & 2(hw + hd) & \le A_{\text{wall}}, \\ & wd & \le A_{\text{floor}}, \\ & \alpha & \le h/w \le \beta, \\ & \gamma & \le d/w \le \delta. \end{array} \tag{7.11}$$

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 (7.11) becomes

$$\begin{array}{ll} \text{maximize} & x + y + z \\ \text{subject to} & \log(\exp(x + y + \log(2/A_{\text{wall}})) + \exp(x + z + \log(2/A_{\text{wall}}))) \le 0, \\ & y + z \le \log(A_{\text{floor}}), \\ & \log(\alpha) \le x - y \le \log(\beta), \\ & \log(\gamma) \le z - y \le \log(\delta). \end{array} \tag{7.12}$$

Next, we demonstrate how to implement a log-sum-exp constraint (7.10). It can be written as:

$$\begin{array}{l} u_k \ge \exp(a_k^T y + b_k), \quad (\text{equiv. } (u_k, 1, a_k^T y + b_k) \in K_{\exp}), \\ \sum_k u_k = 1. \end{array} \tag{7.13}$$

This presentation requires one extra variable u_k for each monomial appearing in the original posynomial constraint. The explicit representation of conic constraints in this case is:

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

We can now use this representation to assemble all constraints in the model.

Listing 7.8: Source code solving problem (7.12).

```
Awall  = 200
Afloor = 50
alpha  = 2
beta   = 10
gamma  = 2
delta  = 10

% A model with variables [x,y,z,u1,u2]
model = mosekmodel(...
```

(continues on next page)

(continued from previous page)

```
        name = "gp1",...
        numvar = 5);
    model.objective("maximize", [ 1 1 1 0 0]);

    % u1 + u2 = 1
    model.appendcons(F = [ 0 0 0 1 1 ], domain = mosekdomain("equal",     rhs=1.0));

    % y + z <= log(Afloor)
    model.appendcons(F = [ 0 1 1 0 0 ], domain = mosekdomain("less than",␣
↪rhs=log(Afloor)));

    % Two-sided bounds on x-y and z-y
    model.appendcons(F = [ 1 -1 0 0 0 ; ...
                          0 -1 1 0 0 ], domain = mosekdomain("greater than",dim=2,␣
↪rhs=[log(alpha) log(gamma)]'));
    model.appendcons(F = [ 1 -1 0 0 0 ; ...
                          0 -1 1 0 0 ], domain = mosekdomain("less than",   dim=2,␣
↪rhs=[log(beta)  log(delta)]'));

    % Conic constraints
    model.appendcons(F = sparse([1 3 3],[4 1 2],[1.0 1.0 1.0]), g = [0 1 log(alpha/
↪Awall)]', domain = mosekdomain("exp"));
    model.appendcons(F = sparse([1 3 3],[5 1 3],[1.0 1.0 1.0]), g = [0 1 log(alpha/
↪Awall)]', domain = mosekdomain("exp"));

    model.solve();
    if model.hassolution("interior")
        [xx,prosta,solsta] = model.getsolution("interior","x");
        x = xx(1:3);
        disp(exp(x));
    end
```

7.4 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.

7.4.1 Example MILO1

We use the example

$$\begin{array}{lccl} \text{maximize} & x_0 + 0.64 x_1 & & \\ \text{subject to} & 50 x_0 + 31 x_1 & \leq & 250, \\ & 3 x_0 - 2 x_1 & \geq & -4, \\ & x_0, x_1 \geq 0 & & \text{and integer} \end{array} \tag{7.14}$$

to demonstrate how to set up and solve a problem with integer variables. It has the structure of a linear optimization problem 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 7.9.

Listing 7.9: How to solve problem (7.14).

```
function [xx,prosta,solsta] = milo1()
    model = mosekmodel(name="milo1", numvar=2, ...
                       intvars=[1 2]');     % Specify indices of integer variables

    model.objective("maximize", [ 1.0 0.64 ]);
    model.appendcons(F = speye(2),    domain = mosekdomain("rplus",dim=2));
    model.appendcons(F = [50.0 31.0], domain = mosekdomain("less than", rhs=250.0));
    model.appendcons(F = [4.0  -2.0], domain = mosekdomain("greater than", rhs=-4));

    model.solve();

    % Access the integer solution
    if model.hassolution("integer")
        [xx,prosta,solsta] = model.getsolution("integer", "x");
        fprintf("Solution: %s\n", sprintf("%g ", xx));
    else
        disp("Solve failed");
    end
end
```

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

- The argument **intvars** is used to specify the indexes of the variables that are integer-constrained.
- The optimal integer solution is fetched using the "integer" solution specifier.

In general, the indices of integer variables can be specified by passing the intvars argument whenever new variables are created in the model, that is either:

- in the function *mosekmodel* when initializing a new model,
- or in the function *mosekmodel.appendvars* when appending new variables to an existing model.

7.4.2 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.

There are two modes for **MOSEK** to utilize an initial solution.

- **A complete solution. MOSEK** will first try to check if the current value of the primal variable solution is a feasible point. The solution can either come from a previous solver call or can be entered by the user, however the full solution with values for all variables (both integer and continuous) must be provided. This check is always performed and does not require any extra action from the user. The outcome of this process can be inspected via information items *"MSK_IINF_MIO_INITIAL_FEASIBLE_SOLUTION"* and *"MSK_DINF_MIO_INITIAL_FEASIBLE_SOLUTION_OBJ"*, and via the Initial feasible solution objective entry in the log.
- **A partial integer solution. MOSEK** can also try to construct a feasible solution by fixing integer variables to the values provided by the user (rounding if necessary) and optimizing over the remaining continuous variables. In this setup the user must provide initial values for all integer variables. This action is only performed if the parameter *MSK_IPAR_MIO_CONSTRUCT_SOL* is switched on. 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.

In the following example we focus on inputting a partial integer solution.

$$\begin{array}{ll} \text{maximize} & 7x_0 + 10x_1 + x_2 + 5x_3 \\ \text{subject to} & x_0 + x_1 + x_2 + x_3 \leq 2.5 \\ & x_0, x_1, x_2 \in \mathbb{Z} \\ & x_0, x_1, x_2, x_3 \geq 0 \end{array} \tag{7.15}$$

Solution values can be set using the method *mosekmodel.setsolution* or via the `solution_x` argument of *mosekmodel*.

Listing 7.10: Implementation of problem (7.15) specifying an initial solution.

```
    % Specify start guess for the integer variables.
    model.setsolution("x", [1 1 0 nan]);

    % Request constructing the solution from integer variable values
    model.solve(param = ["MSK_IPAR_MIO_CONSTRUCT_SOL", "1"]);
```

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 7.11: Retrieving information about usage of initial solution

```
        fprintf("Construct solution used?      %d\n", model.info.MSK_IINF_MIO_
↪CONSTRUCT_SOLUTION)
        fprintf("Construct solution objective: %f\n", model.info.MSK_DINF_MIO_
↪CONSTRUCT_SOLUTION_OBJ);
```

7.4.3 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:

$$
\begin{array}{ll}
\text{minimize} & x^2 + y^2 \\
\text{subject to} & x \geq e^y + 3.8, \\
 & x, y \text{ integer.}
\end{array}
\tag{7.16}
$$

The canonical conic formulation of (7.16) suitable for API for MATLAB is

$$
\begin{array}{lll}
\text{minimize} & t & \\
\text{subject to} & (t, x, y) \in \mathcal{Q}^3 & (t \geq \sqrt{x^2 + y^2}) \\
 & (x - 3.8, 1, y) \in K_{\exp} & (x - 3.8 \geq e^y) \\
 & x, y \text{ integer}, & \\
 & t \in \mathbb{R}. &
\end{array}
\tag{7.17}
$$

Listing 7.12: Implementation of problem (7.17).

```
    %The full variable is [t; x; y]
    model = mosekmodel(name = "mi-conic", ...
                       numvar = 3, objective = [1 0 0], objsense = "min", ...
                       F = [ speye(3); ...
                             0 1 0;    ...
                             0 0 0;    ...
                             0 0 1 ],  ...
                       g = [0 0 0 -3.8 1 0], ...
                       domain = [mosekdomain("qcone", dim = 3), ...
                                 mosekdomain("exp")], ...
                       intvars = [2, 3]);      % Specify the indices of integer␣
↪variables
```

(continues on next page)

```
% It is as always possible (but not required) to input an initial solution
% to start the mixed-integer solver.
model.setsolution("x", [100, 9, -1]);

model.solve();
x = model.getsolution("integer", "x");
fprintf("Optimal (x,y) = (%.2e, %.2e)\n", x(2), x(3));
```

Error and solution status handling were omitted for readability.

7.5 Disjunctive constraints

A **disjunctive constraint (DJC)** involves of a number of affine conditions combined with the logical operators or ($\vee$) and optionally and ($\wedge$) into a formula in *disjunctive normal form*, that is a disjunction of conjunctions. Specifically, a disjunctive constraint has the form of a disjunction

$$T_1 \text{ or } T_2 \text{ or} \cdots \text{ or } T_t \tag{7.18}$$

where each T_i is written as a conjunction

$$T_i = T_{i,1} \text{ and } T_{i,2} \text{ and} \cdots \text{ and } T_{i,s_i} \tag{7.19}$$

and each $T_{i,j}$ is an affine condition (affine equation or affine inequality) of the form $D_{ij}x + d_{ij} \in \mathcal{D}_{ij}$ with $\mathcal{D}_{ij}$ being one of the affine domains from Sec. 13.8.1. A disjunctive constraint (DJC) can therefore be succinctly written as

$$\bigvee_{i=1}^{t} \bigwedge_{j=1}^{s_i} T_{i,j} \tag{7.20}$$

where each $T_{i,j}$ is an affine condition.

Each T_i is called a **term** or **clause** of the disjunctive constraint and t is the number of terms. Each condition $T_{i,j}$ is called a **simple term** and s_i is called the **size** of the i-th term.

A disjunctive constraint is satisfied if at least one of its terms (clauses) is satisfied. A term (clause) is satisfied if all of its constituent simple terms are satisfied. A problem containing DJCs will be solved by the mixed-integer optimizer.

Note that nonlinear cones are not allowed as one of the domains $\mathcal{D}_{ij}$ inside a DJC.

7.5.1 Applications

Disjunctive constraints are a convenient and expressive syntactical tool. Then can be used to phrase many constructions appearing especially in mixed-integer modelling. Here are some examples.

- **Complementarity.** The condition $xy = 0$, where x, y are scalar variables, is equivalent to

  $$x = 0 \text{ or } y = 0.$$

  It is a DJC with two terms, each of size 1.

- **Semicontinuous variable.** A semicontinuous variable is a scalar variable which takes values in $\{0\} \cup [a, +\infty]$. This can be expressed as

  $$x = 0 \text{ or } x \geq a.$$

  It is again a DJC with two terms, each of size 1.

- **Exact absolute value.** The constraint $t = |x|$ is not convex, but can be written as

  $$(x \geq 0 \text{ and } t = x) \text{ or } (x \leq 0 \text{ and } t = -x)$$

  It is a DJC with two terms, each of size 2.

- **Indicator.** Suppose z is a Boolean variable. Then we can write the indicator constraint $z = 1 \implies a^T x \le b$ as

$$(z = 1 \text{ and } a^T x \le b) \text{ or } (z = 0)$$

which is a DJC with two terms, of sizes, respectively, 2 and 1.

- **Piecewise linear functions.** Suppose $a_1 \le \cdots \le a_{k+1}$ and $f : [a_1, a_{k+1}] \to \mathbb{R}$ is a piecewise linear function, given on the i-th of k intervals $[a_i, a_{i+1}]$ by a different affine expression $f_i(x)$. Then we can write the constraint $y = f(x)$ as

$$\bigvee_{i=1}^{k} (a_i \le y \text{ and } y \le a_{i+1} \text{ and } y - f_i(x) = 0)$$

making it a DJC with k terms, each of size 3.

On the other hand most DJCs are equivalent to a mixed-integer linear program through a big-M reformulation. In some cases, when a suitable big-M is known to the user, writing such a formulation directly may be more efficient than formulating the problem as a DJC. See Sec. 12.4.5 for a discussion of this topic.

Disjunctive constraints can be added to any problem which includes linear constraints, affine conic constraints (without semidefinite domains) or integer variables.

7.5.2 Example DJC1

In this tutorial we will consider the following sample demonstration problem:

$$\begin{array}{ll} \text{minimize} & 2x_0 + x_1 + 3x_2 + x_3 \\ \text{subject to} & x_0 + x_1 + x_2 + x_3 \ge -10, \\ & \left(\begin{array}{c} x_0 - 2x_1 \le -1 \\ \text{and} \\ x_2 = x_3 = 0 \end{array}\right) \text{ or } \left(\begin{array}{c} x_2 - 3x_3 \le -2 \\ \text{and} \\ x_0 = x_1 = 0 \end{array}\right), \\ & x_i = 2.5 \text{ for at least one } i \in \{0,1,2,3\}. \end{array} \tag{7.21}$$

The problem has two DJCs: the first one has 2 terms. The second one, which we can write as $\bigvee_{i=0}^{3}(x_i = 2.5)$, has 4 terms (clauses).

We refer to the basic tutorials for the details of constructing a model and setting up variables and linear constraints. In this tutorial we focus on the two disjunctive constraints. Each clause appearing in a disjunction is created with the method *mosekmodel.clause*. A clause is specified similarly to the specification of an ordinary constraint in *mosekmodel.appendcons*, that is in the form

$$Fx + g \in \mathcal{D}$$

where $\mathcal{D}$ is some domain or product of domains.

Therefore the first disjunction in our example can be written as

```
    % A disjunction of two clauses
    model.appenddisjunction( [ model.clause(F = [1 -2 0 0 ;
                                                 0  0 1 0 ;
                                                 0  0 0 1], ...
                                           domain=[ mosekdomain("less than", rhs = [-
↪1]), ...                % 1st simple term of 1st clause
                                                    mosekdomain("equal",     dim = 2,
↪ rhs = [0 0]') ]), ...  % 2nd simple term of 1st clause

                               model.clause(F = [0 0 1 -3 ;
                                                 1 0 0  0 ;
                                                 0 1 0  0], ...
                                           domain = [ mosekdomain("less than", rhs =␣
```

(continues on next page)

(continued from previous page)

```
↪-2), ...                % 1st simple term of 2nd clause
                                                   mosekdomain("equal", dim = 2,␣
↪rhs = [0 0]') ]) ], ...  % 2nd simple term of 2nd clause
                           name = "first-djc");
```

The disjunctive constraint is added to the model with *mosekmodel.appenddisjunction*. Here we call this method with two clauses, each of which is a conjunction of 2 simple terms (although one could just as well think of each clause as a conjunction of 3 simple terms, one per each row of F; this distinction is purely a mater of convention).

The second disjunctive constraint is created by passing an array of 4 clauses:

```
    % A disjunction of four clauses
    model.appenddisjunction([ model.clause(F = [1 0 0 0], domain = mosekdomain("equal
↪", rhs = 2.5)),...
                              model.clause(F = [0 1 0 0], domain = mosekdomain("equal
↪", rhs = 2.5)),...
                              model.clause(F = [0 0 1 0], domain = mosekdomain("equal
↪", rhs = 2.5)),...
                              model.clause(F = [0 0 0 1], domain = mosekdomain("equal
↪", rhs = 2.5)) ],...
                            name = "second-djc");
```

The complete code constructing and solving the problem (7.21) is shown below.

Listing 7.13: Source code solving problem (7.21).

```
    model = mosekmodel(name = "djc1", ...
                       objsense = "minimize", ...
                       objective = [ 2,1,3,1 ]', ...
                       numvar = 4);

    % A disjunction of two clauses
    model.appenddisjunction( [ model.clause(F = [1 -2 0 0 ;
                                                 0  0 1 0 ;
                                                 0  0 0 1], ...
                                            domain=[ mosekdomain("less than", rhs = [-
↪1]), ...                % 1st simple term of 1st clause
                                                     mosekdomain("equal",     dim = 2,
↪ rhs = [0 0]') ]), ...  % 2nd simple term of 1st clause

                               model.clause(F = [0 0 1 -3 ;
                                                 1 0 0  0 ;
                                                 0 1 0  0], ...
                                            domain = [ mosekdomain("less than", rhs =␣
↪-2), ...                % 1st simple term of 2nd clause
                                                       mosekdomain("equal", dim = 2,␣
↪rhs = [0 0]') ]) ], ...  % 2nd simple term of 2nd clause
                           name = "first-djc");

    % A disjunction of four clauses
    model.appenddisjunction([ model.clause(F = [1 0 0 0], domain = mosekdomain("equal
↪", rhs = 2.5)),...
                              model.clause(F = [0 1 0 0], domain = mosekdomain("equal
↪", rhs = 2.5)),...
                              model.clause(F = [0 0 1 0], domain = mosekdomain("equal
↪", rhs = 2.5)),...
                              model.clause(F = [0 0 0 1], domain = mosekdomain("equal
↪", rhs = 2.5)) ],...
```

(continues on next page)

(continued from previous page)

```
                                name = "second-djc");

    % The standard liner constraint
    model.appendcons(name = "C", F = [1 1 1 1], domain = mosekdomain("greater than",␣
↪rhs = -10));

    model.solve();

    if model.hassolution("integer")
        [xx,prosta,solsta] = model.getsolution("integer","x");
        fprintf("Solution : [%s]\n", sprintf("%g ", xx));
    end
```

7.6 Retrieving infeasibility certificates

When a continuous problem is declared as primal or dual infeasible, **MOSEK** provides a Farkas-type infeasibility certificate. If, as it happens in many cases, the problem is infeasible due to an unintended mistake in the formulation or because some individual constraint is too tight, then it is likely that infeasibility can be isolated to a few linear constraints/bounds that mutually contradict each other. In this case it is easy to identify the source of infeasibility. The tutorial in Sec. 8.3 has instructions on how to deal with this situation and debug it **by hand**. We recommend Sec. 8.3 as an introduction to infeasibility certificates and how to deal with infeasibilities in general.

Some users, however, would prefer to obtain the infeasibility certificate using API for MATLAB, for example in order to repair the issue automatically, display the information to the user, or perhaps simply because the infeasibility was one of the intended outcomes that should be analyzed in the code.

In this tutorial we show how to obtain such an infeasibility certificate with API for MATLAB in the most typical case, that is when the linear part of a problem is primal infeasible. A Farkas-type primal infeasibility certificate consists of the dual values of linear constraints and bounds. Each of the dual values (multipliers) indicates that a certain multiple of the corresponding constraint should be taken into account when forming the collection of mutually contradictory equalities/inequalities.

7.6.1 Example PINFEAS

For the purpose of this tutorial we use the same example as in Sec. 8.3, that is the primal infeasible problem

$$
\begin{array}{lccccccccccccccccl}
\text{minimize} & & x_0 & + & 2x_1 & + & 5x_2 & + & 2x_3 & + & x_4 & + & 2x_5 & + & x_6 & & \\
\text{subject to} & s_0: & x_0 & + & x_1 & & & & & & & & & & & \leq & 200, \\
 & s_1: & & & & & x_2 & + & x_3 & & & & & & & \leq & 1000, \\
 & s_2: & & & & & & & & & x_4 & + & x_5 & + & x_6 & \leq & 1000, \\
 & d_0: & x_0 & & & & & & & + & x_4 & & & & & = & 1100, \\
 & d_1: & & & x_1 & & & & & & & & & & & = & 200, \\
 & d_2: & & & & & x_2 & + & & & & & x_5 & & & = & 500, \\
 & d_3: & & & & & & & x_3 & + & & & & & x_6 & = & 500, \\
 & & & & & & & & & & & & & & x_i & \geq & 0.
\end{array} \tag{7.22}
$$

Checking infeasible status and adjusting settings

After the model has been solved we check that it is indeed infeasible. If yes, then we choose a threshold for when a certificate value is considered as an important contributor to infeasibility (ideally we would like to list all nonzero duals, but just like an optimal solution, an infeasibility certificate is also subject to floating-point rounding errors). All these steps are demonstrated in the snippet below:

```
% Check problem status
[hassol, prosta, solsta] = model.hassolution("interior");
if hassol && prosta == "PRIM_INFEAS"
    % Set the tolerance at which we consider a dual value as essential
    eps = 1e-7;
```

Going through the certificate

We now proceed through the dual values and print out the positions of those entries whose dual values exceed the given threshold. These are precisely the values we are interested in:

Listing 7.14: Demonstrates how to retrieve a primal infeasibility certificate.

```
    % Obtain the dual values (contianing certificate)
    y = model.getsolution("interior", "y");

    % List all certificate entries with (sufficiently) nonzero dual values
    disp("Constraint rows participating in infeasibility: ");
    cert = find(abs(y) > eps);
    disp(cert);
```

Running this code will produce the following output:

```
Constraint rows participating in infeasibility:
     1
     3
     4
     5
    13
    14
```

indicating the positions of bounds which appear in the infeasibility certificate with nonzero values.

7.7 The linear/simplex interface tutorial

In this tutorial we demonstrate the linear/simplex optimization part of API for MATLAB. In most cases linear optimization problems can be specified using the main component of API for MATLAB, the conic optimization model; see Sec. 7.1 for an introduction. The separate linear/simplex component documented here is more suitable for users who require:

- using the simplex algorithm,
- warm-starting the simplex algorithm,
- computing a basis solution to a linear problem.

A standard problem in the linear/simplex toolbox is constructed with *moseklinmodel* and has the form

$$
\begin{array}{lrccl}
\text{minimize} & & c^T x + c^f & & \\
\text{subject to} & & Ax & = & b, \\
& b_l \leq & x & \leq & b_u,
\end{array}
\tag{7.23}
$$

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.
- $b \in \mathbb{R}^m$ is the activity of linear constraints.
- $b_l \in \mathbb{R}^n$ is the lower bound for the variables.
- $b_u \in \mathbb{R}^n$ is the upper bound for the variables.

Lower and upper variable bounds can be infinite, or in other words the corresponding bound may be omitted.

A linear problem in this format can be constructed and input either all at once or incrementally, by adding successive blocks of constraints.

It is also possible to extend the model with new variables, modify bounds, input solutions and status keys. etc. This is a topic of a separate tutorial. TODO

7.7.1 Example LO1

We set up a linear problem in one go. Consider the problem

$$\begin{array}{lccccccccl} \text{maximize} & 3x_1 & + & 1x_2 & + & 5x_3 & + & 1x_4 & & \\ \text{subject to} & 3x_1 & + & 1x_2 & + & 2x_3 & & & = & 30, \\ & 2x_1 & + & 1x_2 & + & 3x_3 & + & 1x_4 & \geq & 15, \\ & & & 2x_2 & & & + & 3x_4 & \leq & 25, \end{array} \tag{7.24}$$

with variable bounds

$$\begin{array}{ccccc} 0 & \leq & x_1 & \leq & \infty, \\ 0 & \leq & x_2 & \leq & 10, \\ 0 & \leq & x_3 & \leq & \infty, \\ 0 & \leq & x_4 & \leq & \infty. \end{array}$$

We need to reformulate the problem so that the constraints are of the form $Ax = b$ by introducing nonnegative slack variables:

$$\begin{array}{lcccccccccccl} \text{maximize} & 3x_1 & + & 1x_2 & + & 5x_3 & + & 1x_4 & & & & & \\ \text{subject to} & 3x_1 & + & 1x_2 & + & 2x_3 & & & & & & = & 30, \\ & 2x_1 & + & 1x_2 & + & 3x_3 & + & 1x_4 & - & s_1 & & = & 15, \\ & & & 2x_2 & & & + & 3x_4 & & & + \; s_2 & = & 25, \end{array} \tag{7.25}$$

with variable bounds

$$\begin{array}{ccccc} 0 & \leq & x_1, x_3, x_4, s_1, s_2 & \leq & \infty, \\ 0 & \leq & x_2 & \leq & 10. \end{array}$$

We create a model by providing all data at once:

- the number of variables is 5 and is set as `numvar`,
- the objective vector is $c = [3, 1, 5, 1, 0, 0]$ and is passed as `objective`,
- the objective sense is maximization, and is passed as `objsense`,
- the problem name `name` (optional) is `lo1`,
- we also add variable and constraint names for illustration.

Listing 7.15: Setting up a linear model in a single call.

```
model = moseklinmodel(...
              name = "lo1", ...
              numvar = 6, ...
              objsense = "maximize", ...
              objective = [ 3 1 5 1 0 0 ]', ...
              A = [ 3 1 2 0  0  0 ; ...
                    2 1 3 1 -1  0 ; ...
                    0 2 0 3  0 -1 ], ...
              b = [ 30 15 25 ]', ...
              blx = [ 0.0  0.0 0.0 0.0   0.0 -inf ]', ...
              bux = [ inf 10.0 inf inf   inf  0.0 ]', ...
              varnames = ["x1" "x2" "x3" "x4" "y1" "y2"]',...
              conname = ["c1" "c2" "c3"]');
```

7.7.2 Example with multiple calls (LO2)

We now show how to set up the same model (7.25) adding various elements of the problem in separate calls. For illustration we also split the constraints intwo two blocks and add them separately. The only rule to be observed is that each element can only refer to elements already defined, for example we can not add the objective before defining the number of variables, i.e. the model must remain consistent at all times throughout the process.

Listing 7.16: Setting up a linear model with multiple calls.

```
% Create an empty model
model = moseklinmodel(name = "lo2");

% Add some variables with bounds
model.appendvars(6,...
                 bl = [ 0.0  0.0 0.0 0.0   0.0 -inf ]', ...
                 bu = [ inf 10.0 inf inf   inf  0.0 ]');
model.varname([1:6],["x1" "x2" "x3" "x4" "y1" "y2"]);

% Set the objective
model.objective("maximize", [ 3 1 5 1 0 0]', objfixterm = 0.0);

% Constraint 3x1 + x2 + 2x3 == 30
model.appendcons([ 3 1 2 0 0 0], [30]);

% The two remaining constraints
model.appendcons([ 2 1 3 1 -1  0 ; ...
                   0 2 0 3  0 -1 ], ...
                 [ 15 25 ]');
```

7.7.3 Solving and retrieving the solution

We wrap up with a short demonstration of what to do after the model has been defined, that is how to solve the model and retrieve solutions.

Listing 7.17: Solving and retrieving the solution.

```
% Choose the simplex optimizer to solve with
model.solve(param = [ "MSK_IPAR_OPTIMIZER", "MSK_OPTIMIZER_FREE_SIMPLEX" ]);

% Check the solution status and fetch the solution
[hassol, prosta, solsta] = model.hassolution();
```

(continues on next page)

(continued from previous page)

```
    if hassol && solsta == "OPTIMAL"
        fprintf("Solution, problem status = %s, solution status = %s:\n", prosta,␣
↪solsta);

        % Primal variables
        result = model.getsolution("x");
        x = result(1:4);
        s = result(5:6);
        fprintf("  x[%d] = %.4f\n",[1:4 ; x']);
        fprintf("Slacks:\n");
        fprintf("  s[%d] = %.4f\n",[1:2 ; s']);

        % Other elements of the solution
        fprintf("Duals:\n");
        fprintf("  y[%d] = %.4f\n",[1:3 ; model.getsolution("y")']);
        fprintf("slx[%d] = %.4f\n",[1:6 ; model.getsolution("slx")']);
        fprintf("sux[%d] = %.4f\n",[1:6 ; model.getsolution("sux")']);

        fprintf("Status keys of constraints:\n");
        fprintf("skc[%s] = %s\n",[1:3 ; model.getsolution("skc")']);

        fprintf("Status keys of variables:\n");
        fprintf("skx[%s] = %s\n",[1:6 ; model.getsolution("skx")']);
    end
```

We note that

- The optimizer is invoked with *moseklinmodel.solve*.
- We check if the basic solution is available with *moseklinmodel.hassolution*, which returns also problem and solution status.
- The primal solution is obtained by requesting the "x" component in *moseklinmodel.getsolution*.
- Dual values and status keys are obtained with the repective other calls to *moseklinmodel.getsolution*.
- See Sec. 6.1 for more details about retrieving solutions and handling more solution statuses, and Sec. 6.2 for information about error handling, which we omitted for readability.

7.8 The linear/simplex warm-start tutorial

In this tutorial we demonstrate how to warm-start the simplex optimizer using the linear/simplex optimization part of API for MATLAB. See Sec. 7.7 for a general introduction to specifying linear problems using this tool.

The simplex optimizers can be warm-started to speed up the solution process by providing a good initial point and thereby reducing the number of simplex iterations required to reach the optimal basic solution. The efficiency of warm-start depends on the quality and amount of information provided, typically this will be any of the following:

- a primal feasible point,
- status keys indicating if the variables are basic or on their bounds,
- full primal-dual solution,
- an optimal solution from a previous solve which remains primal or dual feasible after problem modification and can be used to warm-start a subsequent solve.

We demonstrate warm-starting on two examples

- providing the initial information with the problem data,
- reusing a solution from a previous solve after modifying the problem.

In both cases we work with the same sample problem as in Sec. 7.7:

$$\begin{array}{lccccccccccccl} \text{maximize} & 3x_1 & + & 1x_2 & + & 5x_3 & + & 1x_4 & & & & & & \\ \text{subject to} & 3x_1 & + & 1x_2 & + & 2x_3 & & & & & & & = & 30, \\ & 2x_1 & + & 1x_2 & + & 3x_3 & + & 1x_4 & - & s_1 & & & = & 15, \\ & & & 2x_2 & & & + & 3x_4 & & & + & s_2 & = & 25, \end{array} \tag{7.26}$$

with variable bounds

$$\begin{array}{ccccc} 0 & \leq & x_1, x_3, x_4, s_1, s_2 & \leq & \infty, \\ 0 & \leq & x_2 & \leq & 10. \end{array}$$

7.8.1 Inputting warm-start information

If the initial warm-start information is available it can be entered together with the full specification of the problem in the constructor of *moseklinmodel*. Alternatively, the function *moseklinmodel.setsolution* can also be used for that purpose.

Listing 7.18: Setting up a linear model with warm-start.

```
% We set up the linear model
% together with:
% - approximate primal solution
% - status keys
model = moseklinmodel(...
            name = "lo1", ...
            numvar = 6, ...
            objsense = "maximize", ...
            objective = [ 3 1 5 1 0 0 ]', ...
            A = [ 3 1 2 0  0  0 ; ...
                  2 1 3 1 -1  0 ; ...
                  0 2 0 3  0 -1 ], ...
            b = [ 30 15 25 ]', ...
            blx = [ 0.0  0.0 0.0 0.0   0.0 -inf ]', ...
            bux = [ inf 10.0 inf inf   inf  0.0 ]', ...
            varnames = ["x1" "x2" "x3" "x4" "y1" "y2"]',...
            conname = ["c1" "c2" "c3"]', ...
            solution_x = [0 0 15 8.3 0 0]', ...
            solution_skx = ["LOW"  "LOW"  "BAS"  "BAS"  "BAS"  "UPR"]', ...
            solution_skc = ["FIX"  "FIX", "FIX"]);
```

In this example we input a primal solution (approximate) and status keys as defined in *stakey*. Additional data that can be entered at this point is the dual solution: `solution_y`, `solution_slx` and `solution_sux`.

The simplex optimizer (primal, dual or free) is then invoked and solution retrieved as in Sec. 7.7.

We can check solver statistics to validate that the initial solution was taken into account and whether it reduced the number of simplex iterations:

Listing 7.19: Obtaining solver statistics.

```
% Solver statistics
fprintf("Optimizer time %.3f\n", model.info.MSK_DINF_OPTIMIZER_TIME);
fprintf("#iterations    %ld\n",  model.info.MSK_LIINF_SIMPLEX_ITER);
```

7.8.2 Warm-starting after modification

In this scenario we solve a sequence of linear problems obtained by modifying the data in a single *moseklinmodel* object. In this case the solution from a previous solve remains in the object and will be used to initialize a subsequent solve without any extra action form the user.

Suppose first we solve some initial problem:

Listing 7.20: Defining and optimizing some initial model.

```
% We set up and initially solve a sample linear model
model = moseklinmodel(...
              numvar = 6, ...
              objsense = "maximize", ...
              objective = [ 3 1 5 1 0 0 ]', ...
              A = [ 3 1 2 0  0  0 ; ...
                    2 1 3 1 -1  0 ; ...
                    0 2 0 3  0 -1 ], ...
              b = [ 30 15 25 ]', ...
              blx = [ 0.0  0.0 0.0 0.0   0.0 -inf ]', ...
              bux = [ inf 10.0 inf inf   inf  0.0 ]');

model.solve(param = [ "MSK_IPAR_OPTIMIZER", "MSK_OPTIMIZER_FREE_SIMPLEX" ], quiet␣
↪= true);
```

Later, we may want to change an upper bound (make it more strict) using *moseklinmodel.setb*. This operation preserves dual feasibility of the previous solution stored in the model, so it makes sense to solve the new problem with the dual simplex. The previous solution will be fed as an initial warm start to the solver automatically:

Listing 7.21: Updating a bound and reoptimizing.

```
% Introduce a stricter upper bound for the 3-rd constraint
model.setb([30, 15, 22]');

% Solve, using dual simplex
model.solve(param = [ "MSK_IPAR_OPTIMIZER", "MSK_OPTIMIZER_DUAL_SIMPLEX" ]);
```

Similarly, we may now want to change the objective vector using *moseklinmodel.setc*. This operation preserves primal feasibility of the previous solution stored in the model, so it makes sense to solve the new problem with the primal simplex. The previous solution will be fed as an initial warm start to the solver automatically:

Listing 7.22: Updating the objective and reoptimizing.

```
% Change an objective coefficient
model.setc([3 2 0 2]');

% Solve, using primal simplex
model.solve(param = [ "MSK_IPAR_OPTIMIZER", "MSK_OPTIMIZER_PRIMAL_SIMPLEX" ]);
```

The solution from a previous solve is propagated to the next solve of the same *moseklinmodel*, unless the user inputs any new element of the initial solution using *moseklinmodel.setsolution*, at which point that solution is used to warm-start the immediately following solve. This pattern continues.

For a more in-depth treatment see the following sections:

- Sec. 9 for more advanced and complicated optimization examples.
- Sec. 9.1 for examples related to portfolio optimization.
- Sec. 11 for formal mathematical formulations of problems **MOSEK** can solve, dual problems and infeasibility certificates.

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:

```
model.solve(quiet = false);
```

- 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. 6.3.3.
  - use a human-readable text format, preferably `*.ptf` if you want to check the problem structure by hand. Assign names to variables and constraints to make them easier to identify.

```
% Before solving:
model.solve(write_to_file = "dump.ptf");
% or without solving:
model.write("dump.ptf");
```

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

```
% Before solving:
model.solve(write_to_file = "dump.task.gz");
% or without solving:
model.write("dump.task.gz");
```

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

See the following sections for details.

8.1 Understanding optimizer log

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

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

In this tutorial we show how to analyze the most important parts of the log when initially debugging a model: input data (1) and solution summary (4). For the iterations log (3) see Sec. 12.3.4 or Sec. 12.4.3.

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        : minimize
  Type                   : CONIC (conic optimization problem)
  Constraints            : 234
  Affine conic cons.     : 5348 (6444 rows)
  Disjunctive cons.      : 0
  Cones                  : 0
  Scalar variables       : 20693
  Matrix variables       : 1 (scalarized: 45)
  Integer variables      : 0
```

This can be consulted to eliminate simple errors: wrong objective sense, wrong number of variables etc. Note that some modeling tools can introduce additional variables and constraints to the model and perturb the model even further (such as by dualizing). In most **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

** Affine conic constraints (ACC)
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
```

(continues on next page)

(continued from previous page)

```
|F|                 nnz: 612301        min=8.29e-06        max=9.31e+01
|g|                 nnz: 1203          min=5.00e-03        max=1.00e+00
```

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

- Wrong type of conic constraint 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 PTF (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.

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    ␣
↪acc: 3e-11
Dual.    obj: 8.7560521345e+01    nrm: 1e+00    Viol.  con: 5e-09    var: 9e-11    ␣
↪acc: 0e+00
```

It contains the following elements:

- Problem and solution status.
- A summary of the primal solution: objective value, infinity norm of the solution vector and maximal violations of variables and constraints of different types. 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 is the distance to the cone.
- 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  ␣
↪acc: 0e+00
Dual.    obj: 3.0119004098e-01    nrm: 5e+07    Viol.  con: 4e-16    var: 1e-01  ␣
↪acc: 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. 12.3.2). Consider for instance:

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

It is a Farkas-type certificate as described in Sec. 11.1.2. 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   ␣
↪acc: 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. 12.4 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.

Some information about the numerical properties of the data 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.

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. 12.1.
- 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:

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

If we add the equalities together we obtain:

$$0 = \varepsilon$$

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

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

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

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

When solving an optimization problem one typically expects to get an optimal solution, but in some cases, either by design, or (most frequently) due to an error in the formulation, the problem may become infeasible (have no solution at all).

This section

- describes the intuitions behind infeasibility,
- helps to debug (unexpectedly) infeasible problems using the command line tool and by inspecting infeasibility reports and problem data by hand,
- gives some hints for how to modify the formulation to identify the reasons for infeasibility.

If, instead, you want to fetch an infeasibility certificate directly using API for MATLAB, see the tutorial in Sec. 7.6.

An infeasibility certificate is only available for continuous problems, however the hints in Sec. 8.3.4 apply to a large extent also to mixed-integer problems.

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.

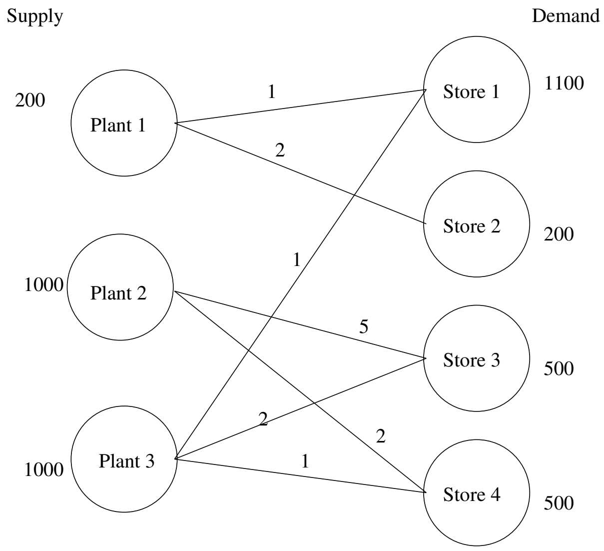

Fig. 8.1: Supply, demand and cost of transportation.

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:

$$
\begin{array}{lllllllllllllllll}
\text{minimize} & & x_{11} & + & 2x_{12} & + & 5x_{23} & + & 2x_{24} & + & x_{31} & + & 2x_{33} & + & x_{34} & & \\
\text{subject to} & s_0 : & x_{11} & + & x_{12} & & & & & & & & & & & \leq & 200, \\
 & s_1 : & & & & & x_{23} & + & x_{24} & & & & & & & \leq & 1000, \\
 & s_2 : & & & & & & & & & x_{31} & + & x_{33} & + & x_{34} & \leq & 1000, \\
 & d_0 : & x_{11} & & & & & & & + & x_{31} & & & & & = & 1100, \\
 & d_1 : & & & x_{12} & & & & & & & & & & & = & 200, \\
 & d_2 : & & & & & x_{23} & + & & & & & x_{33} & & & = & 500, \\
 & d_3 : & & & & & & & x_{24} & + & & & & & x_{34} & = & 500, \\
 & & & & & & & & & & & & & & x_{ij} & \geq & 0.
\end{array}
\tag{8.1}
$$

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. 11.2.2 or Sec. 11.1.2 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

```
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                200                 0                   1
2             s2        none                1000                0                   1
3             d0        1100                1100                1                   0
4             d1        200                 200                 1                   0


The following bound constraints are involved in the primal infeasibility.


Index         Name      Lower bound         Upper bound         Dual lower          Dual upper
5             x33       0                   none                1                   0
6             x34       0                   none                1                   0
```

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`, `d0`, `d1` 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. 11.2.2).

In our example the certificate means that an appropriate linear combination of constraints `s0`, `s1` with coefficient $s_u^c = 1$, constraints `d0` and `d1` 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:

```
Dual infeasibility report

Problem status: The problem is dual infeasible


The following constraints are involved in the dual infeasibility.


Index         Name      Activity          Objective         Lower bound       Upper bound
5             x33       -1                                  none              2
6             x34       -1                                  none              1


The following variables are involved in the dual infeasibility.


Index         Name      Activity          Objective         Lower bound       Upper bound
0             y1        -1                200               none              0
2             y3        -1                1000              none              0
3             y4        1                 1100              none              none
4             y5        1                 200               none              none
```

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. 11.2.2. 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 PTF 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 PTF 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 3. The **MOSEK** interface for Python must be installed following the installation instructions for Python API or Python Fusion API. The easiest option is

```
pip install Mosek
```

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

To run the console in interactive mode use

```
python mosekconsole.py
```

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

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

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

8.4.2 Examples

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

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

To convert between file formats:

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

To set a parameter before solving:

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

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

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

To print a summary of problem structure:

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

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

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

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

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

8.4.3 Full list of commands

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

```
help cmd
```

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

Command	Description
`help [command]`	Print list of commands or info about a specific command
`log filename`	Save the session to a file
`intro`	Print MOSEK splashscreen
`testlic`	Test the license system
`read filename`	Load problem from file
`reread`	Reload last problem file
`solve [options]`	Solve current problem
`write filename`	Write current problem to file
`param [name [value]]`	Set a parameter or get parameter values
`paramdef`	Set all parameters to default values
`paramdiff`	Show parameters with non-default values
`paramval name`	Show available values for a parameter
`info [name]`	Get an information item
`anapro`	Analyze problem data
`anapro+`	Analyze problem data with the internal analyzer
`hist`	Plot a histogram of problem data
`histsol`	Plot a histogram of the solutions
`spy`	Plot the sparsity pattern of the data matrices
`truncate epsilon`	Truncate small coefficients down to 0
`resobj [fac]`	Rescale objective by a factor
`rlb thr`	Remove large bounds
`anasol`	Analyze solutions
`removeitg`	Remove integrality constraints
`removecones`	Remove all cones and leave just the linear part
`delsol`	Remove solutions
`fixsol solname`	Fix all variables to a specific solution
`fixintsol`	Fix all integer variables to a specific solution
`infsub`	Replace current problem with its infeasible subproblem
`dualize`	Replace current problem with its dual
`writesol basename`	Write solution(s) to file(s) with given basename

continues on next page

Table 8.1 – continued from previous page

Command	Description
`writejsonsol name`	Write solutions to JSON file with given name
`ptf`	Print the PTF representation of the problem
`optserver [url]`	Use an OptServer to optimize
`ls`	List the current folder
`exit`	Leave

Chapter 9

Case Studies

In this section we present some case studies in which the API 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. 7 before going through these advanced case studies.

- *Portfolio Optimization*
  - **Keywords:** Markowitz model, variance, risk, efficient frontier, factor model, 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

9.1 Portfolio Optimization

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

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

9.1.1 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

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

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

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

Since e is the vector of all ones then

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

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

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

This leads to the first constraint

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

The second constraint

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

ensures that the variance, 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 $G \in \mathbb{R}^{n \times k}$ such that

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

In general the choice of G is **not** unique and one possible choice of G is the Cholesky factorization of Σ. However, in many cases another choice is better for efficiency reasons as discussed in Sec. 9.1.3. For a given G we have that

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

Hence, we may write the risk constraint as

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

or equivalently

$$(\gamma, G^T x) \in \mathcal{Q}^{k+1},$$

where $\mathcal{Q}^{k+1}$ is the $(k+1)$-dimensional quadratic cone. Note that specifically when G is derived using Cholesky factorization, $k = n$.

Therefore, problem (9.1) can be written as

$$\begin{array}{lrcl} \text{maximize} & \mu^T x & & \\ \text{subject to} & e^T x & = & w + e^T x^0, \\ & (\gamma, G^T x) & \in & \mathcal{Q}^{k+1}, \\ & x & \geq & 0, \end{array} \tag{9.3}$$

which is a conic quadratic optimization problem that can easily be formulated and solved with API for MATLAB. Subsequently we will use the example data

$$\mu = \left[\begin{array}{c} 0.0720, 0.1552, 0.1754, 0.0898, 0.4290, 0.3929, 0.3217, 0.1838 \end{array} \right]^T$$

and

$$\Sigma = \left[\begin{array}{cccccccc} 0.0946 & 0.0374 & 0.0349 & 0.0348 & 0.0542 & 0.0368 & 0.0321 & 0.0327 \\ 0.0374 & 0.0775 & 0.0387 & 0.0367 & 0.0382 & 0.0363 & 0.0356 & 0.0342 \\ 0.0349 & 0.0387 & 0.0624 & 0.0336 & 0.0395 & 0.0369 & 0.0338 & 0.0243 \\ 0.0348 & 0.0367 & 0.0336 & 0.0682 & 0.0402 & 0.0335 & 0.0436 & 0.0371 \\ 0.0542 & 0.0382 & 0.0395 & 0.0402 & 0.1724 & 0.0789 & 0.0700 & 0.0501 \\ 0.0368 & 0.0363 & 0.0369 & 0.0335 & 0.0789 & 0.0909 & 0.0536 & 0.0449 \\ 0.0321 & 0.0356 & 0.0338 & 0.0436 & 0.0700 & 0.0536 & 0.0965 & 0.0442 \\ 0.0327 & 0.0342 & 0.0243 & 0.0371 & 0.0501 & 0.0449 & 0.0442 & 0.0816 \end{array} \right].$$

Using Cholesky factorization, this implies

$$G^T = \left[\begin{array}{cccccccc} 0.3076 & 0.1215 & 0.1134 & 0.1133 & 0.1763 & 0.1197 & 0.1044 & 0.1064 \\ 0. & 0.2504 & 0.0995 & 0.0916 & 0.0669 & 0.0871 & 0.0917 & 0.0851 \\ 0. & 0. & 0.1991 & 0.0587 & 0.0645 & 0.0737 & 0.0647 & 0.0191 \\ 0. & 0. & 0. & 0.2088 & 0.0493 & 0.0365 & 0.0938 & 0.0774 \\ 0. & 0. & 0. & 0. & 0.3609 & 0.1257 & 0.1016 & 0.0571 \\ 0. & 0. & 0. & 0. & 0. & 0.2155 & 0.0566 & 0.0619 \\ 0. & 0. & 0. & 0. & 0. & 0. & 0.2251 & 0.0333 \\ 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0.2202 \end{array} \right]$$

In Sec. 9.1.3, we present a different way of obtaining G based on a factor model, that leads to more efficient computation.

Why a Conic Formulation?

Problem (9.1) is a convex quadratically constrained optimization problem that can be solved directly using **MOSEK**. Why then reformulate it as a conic quadratic optimization problem (9.3)? 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 (9.1) 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

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

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 9.1 demonstrates how the basic Markowitz model (9.3) is implemented.

Listing 9.1: Code implementing problem (9.3).

```
model = mosekmodel(...
    name="Basic Markowitz", ...
    numvar = n, ...
    objsense = "maximize", ...
    objective = mu);

% Bound on the variables (holdings). Shortselling is not allowed.
model.appendcons(F = eye(n), domain = mosekdomain("nonnegative", n = n));

% The amount invested  must be identical to initial wealth
model.appendcons(F = ones(1, n), domain = mosekdomain("equal", rhs=w+sum(x0)));

% Imposes a bound on the risk
model.appendcons(F = [zeros(1,n); GT], g = [gamma; zeros(m,1)],...
                 domain = mosekdomain("qcone", dim = m+1));

model.solve();

% Check if the interior point solution is an optimal point
% See https://docs.mosek.com/latest/matlabapi/accessing-solution.html about␣
↪handling solution statuses.
[ok,prosta,solsta] = model.hassolution("interior");
if ~ ok || solsta ~= "OPTIMAL"
    error("Unexpected solution status");
else
    xx = model.getsolution("interior","x");

    disp("\n-----------------------------------------------------------------------
↪--------------");
    disp("Basic Markowitz portfolio optimization");
    disp("-------------------------------------------------------------------------
↪------------\n");
    fprintf("Expected return: %.4e Std. deviation: %.4e\n", mu' * xx, gamma);
end
```

The source code should be self-explanatory except perhaps for

```
model.appendcons(F = [zeros(1,n); GT], g = [gamma; zeros(m,1)],...
                 domain = mosekdomain("qcone", dim = m+1));
```

where the constraint

$$(\gamma, G^T x) \in \mathcal{Q}^{k+1}$$

is created as an *affine conic constraint format* of the form $Fx + g \in \mathcal{K}$, in this specific case:

$$\begin{bmatrix} 0 \\ G^T \end{bmatrix} x + \begin{bmatrix} \gamma \\ 0 \end{bmatrix} \in \mathcal{Q}^{k+1}.$$

9.1.2 The Efficient Frontier

The portfolio computed by the Markowitz model is efficient in the sense that there is no other portfolio giving a strictly higher return for the same amount of risk. An efficient portfolio is also sometimes called a Pareto optimal portfolio. Clearly, an investor should only invest in efficient portfolios and therefore it may be relevant to present the investor with all efficient portfolios so the investor can choose the portfolio that has the desired tradeoff between return and risk.

Given a nonnegative α the problem

$$\begin{array}{ll} \text{maximize} & \mu^T x - \alpha x^T \Sigma x \\ \text{subject to} & e^T x = w + e^T x^0, \\ & x \geq 0. \end{array} \tag{9.4}$$

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:

$$\begin{array}{llll} \text{maximize} & \mu^T x - \alpha s & & \\ \text{subject to} & e^T x & = & w + e^T x^0, \\ & (s, 0.5, G^T x) & \in & Q_r^{k+2} \quad (\text{equiv. to } s \geq \|G^T x\|_2^2 = x^T \Sigma x), \\ & x & \geq & 0. \end{array} \tag{9.5}$$

The parameter α specifies the tradeoff between expected return and variance. Ideally the problem (9.4) should be solved for all values $\alpha \geq 0$ but in practice it is impossible. Using the example data from Sec. 9.1.1, the optimal values of return and variance for several values of α are shown in the figure.

Example code

Listing 9.2 demonstrates how to compute the efficient portfolios for several values of α.

Listing 9.2: Code for the computation of the efficient frontier based on problem (9.4).

```
    for i = 1:niter
        alpha = alphas(i);
        model = mosekmodel(name = "Efficient frontier", numvar = n+1);

        % Defines the variables (holdings). Shortselling is not allowed.
        model.appendcons(F = speye(n), domain = mosekdomain("greater than", dim = n,␣
↪rhs = zeros(n,1)));

        x_idxs = [1:n];
        s_idxs = n+1;

        model.appendcons(name = "budget", F = [ones(1,n) 0], domain = mosekdomain(
↪"equal", rhs = w + sum(x0)));

        % Computes the risk
        model.appendcons(name="variance", ...
                         F=[ sparse(1,n+1,[1.0],[1],[s_idxs]) ; ...
                             sparse(1,n+1) ; ...
                             GT zeros(n,1)], ...
                         g=[0; 0.5; zeros(n,1)], ...
                         domain = mosekdomain("rqcone", dim = n+2));
```

(continues on next page)

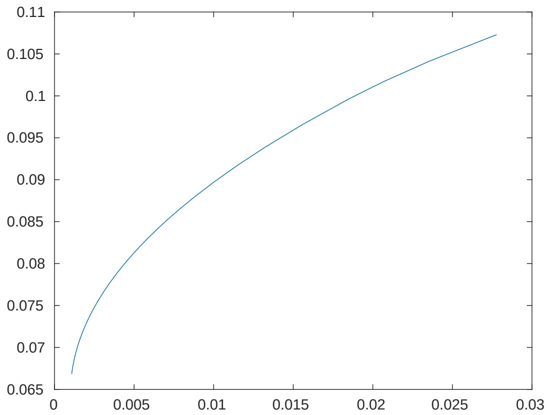

Fig. 9.1: The efficient frontier for the sample data.

(continued from previous page)

```
        %  Define objective as a weighted combination of return and variance
        model.objective("maximize", [ mu ; -alpha ]);

        % Solve the problem for the current alpha
        model.solve();

        % Check if the solution is an optimal point
        % See https://docs.mosek.com/latest/matlabapi/accessing-solution.html about␣
↪handling solution statuses.
        [ok,prosta,solsta] = model.hassolution("interior");
        if ~ ok || solsta ~= "OPTIMAL"
            error("Unexpected solution status");
        else
            xx = model.getsolution("interior","x");
            x = xx(x_idxs,1);
            s = xx(s_idxs,1);

            frontier_mux(i,1) = mu' * x;
            frontier_s(i,1)   = s;
        end
    end

    fprintf("\n-----------------------------------------------------------------------------
↪------------\n");
    fprintf("Efficient frontier") ;
    fprintf("\n-----------------------------------------------------------------------------
↪------------\n");
    fprintf("\t%-12s  %-12s  %-12s\n", "alpha", "return", "std. dev.");

    sqrt_frontier_s = sqrt(frontier_s);
    for i = 1:niter
        fprintf("\t%-12.4f  %-12.4e  %-12.4e\n", alphas(i,1), frontier_mux(i,1),␣
↪frontier_s(i,1));
    end
```

9.1.3 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 (9.2) and try to reduce that number by for instance changing the choice of G.

In other words if the computational efficiency should be improved then it is always good idea to start with focusing at the covariance matrix. As an example assume that

$$\Sigma = D + VV^T$$

where D is a positive definite diagonal matrix. Moreover, V is a matrix with n rows and k columns. Such a model for the covariance matrix is called a factor model and usually k is much smaller than n. In practice k 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

$$G = \left[\begin{array}{cc} D^{1/2} & V \end{array} \right]$$

because then

$$GG^T = D + VV^T.$$

This choice requires storage proportional to $n + kn$ which is much less than for the Cholesky choice of G. Indeed assuming k 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].

Factor model in finance

Factor model structure is typical in financial context. It is common to model security returns as the sum of two components using a factor model. The first component is the linear combination of a small number of factors common among a group of securities. The second component is a residual, specific to each security. It can be written as $R = \sum_j \beta_j F_j + \theta$, where R is a random variable representing the return of a security at a particular point in time, F_j is the random variable representing the common factor j, β_j is the exposure of the return to factor j, and θ is the specific component.

Such a model will result in the covariance structure

$$\Sigma = \Sigma_\theta + \beta \Sigma_F \beta^T,$$

where Σ_F is the covariance of the factors and Σ_θ is the residual covariance. This structure is of the form discussed earlier with $D = \Sigma_\theta$ and $V = \beta P$, assuming the decomposition $\Sigma_F = PP^T$. If the number of factors k is low and Σ_θ is diagonal, we get a very sparse G that provides the storage and solution time benefits.

Example code

Here we will work with the example data of a two-factor model ($k = 2$) built using the variables

$$\beta = \begin{bmatrix} 0.4256 & 0.1869 \\ 0.2413 & 0.3877 \\ 0.2235 & 0.3697 \\ 0.1503 & 0.4612 \\ 1.5325 & -0.2633 \\ 1.2741 & -0.2613 \\ 0.6939 & 0.2372 \\ 0.5425 & 0.2116 \end{bmatrix},$$

$$\theta = [0.0720, 0.0508, 0.0377, 0.0394, 0.0663, 0.0224, 0.0417, 0.0459],$$

and the factor covariance matrix is

$$\Sigma_F = \begin{bmatrix} 0.0620 & 0.0577 \\ 0.0577 & 0.0908 \end{bmatrix},$$

giving

$$P = \begin{bmatrix} 0.2491 & 0. \\ 0.2316 & 0.1928 \end{bmatrix}.$$

Then the matrix G would look like

$$G = \begin{bmatrix} \beta P & \Sigma_\theta^{1/2} \end{bmatrix} = \begin{bmatrix} 0.1493 & 0.0360 & 0.2683 & 0. & 0. & 0. & 0. & 0. & 0. & 0. \\ 0.1499 & 0.0747 & 0. & 0.2254 & 0. & 0. & 0. & 0. & 0. & 0. \\ 0.1413 & 0.0713 & 0. & 0. & 0.1942 & 0. & 0. & 0. & 0. & 0. \\ 0.1442 & 0.0889 & 0. & 0. & 0. & 0.1985 & 0. & 0. & 0. & 0. \\ 0.3207 & -0.0508 & 0. & 0. & 0. & 0. & 0.2576 & 0. & 0. & 0. \\ 0.2568 & -0.0504 & 0. & 0. & 0. & 0. & 0. & 0.1497 & 0. & 0. \\ 0.2277 & 0.0457 & 0. & 0. & 0. & 0. & 0. & 0. & 0.2042 & 0. \\ 0.1841 & 0.0408 & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0.2142 \end{bmatrix}.$$

This matrix is indeed very sparse.

In general, we get an $n \times (n+k)$ size matrix this way with k full columns and an $n \times n$ diagonal part. In order to maintain a sparse representation we do not construct the matrix G explicitly in the code but instead work with two pieces of data: the dense matrix $G_{\text{factor}} = \beta P$ of shape $n \times k$ and the diagonal vector θ of length n.

Example code

In the following we demonstrate how to write code to compute the matrix G_{factor} of the factor model. We start with the inputs

Listing 9.3: Inputs for the computation of the matrix G_{factor} from the factor model.

```
B    = [ 0.4256, 0.1869; ...
         0.2413, 0.3877; ...
         0.2235, 0.3697; ...
         0.1503, 0.4612; ...
         1.5325, -0.2633; ...
         1.2741, -0.2613; ...
         0.6939, 0.2372; ...
         0.5425, 0.2116 ];

S_F = [0.0620, 0.0577; ...
       0.0577, 0.0908 ];

theta = [0.0720 0.0508 0.0377 0.0394 0.0663 0.0224 0.0417 0.0459];
```

Then the matrix G_{factor} is obtained as:

```
P   = chol(S_F)';
G_factor   = B * P;
G_factor_T = G_factor';
```

The code for computing an optimal portfolio in the factor model is very similar to the one from the basic model in Listing 9.1 with one notable exception: we construct the expression $G^T x$ appearing in the conic constraint by stacking together two separate vectors $G_{\text{factor}}^T x$ and $\Sigma_\theta^{1/2} x$:

```
    model.appendcons(name = "risk", ...
                     F = sparse([ zeros(1,n); G_factor_T; diag(sqrt(theta)) ]), ..
↪.
                     g = [gamma; zeros(size(G_factor_T, 1) + n, 1)], ...
                     domain = mosekdomain("qcone", dim = size(G_factor_T, 1) + n␣
↪+ 1 ));
```

The full code is demonstrated below:

Listing 9.4: Implementation of portfolio optimization in the factor model.

```
    % A model with variable x and maximized return
    model = mosekmodel(name = "Factor risk", ...
                       numvar = n, ...
                       objsense = "maximize", ...
                       objective = mu);

    % Linear budget constraint
    model.appendcons(name = "budget", F = ones(1,n), ...
                     domain = mosekdomain("equal", rhs = w + sum(x0)));

```

(continues on next page)

(continued from previous page)

```
        % No shortselling, x >= 0
        model.appendcons(F = speye(n),  ...
                         domain = mosekdomain("greater than", dim = n, rhs = zeros(n,
↪1)));

        % An affine conic constraint: [gamma, G_factor_T*x, sqrt(theta).'x ] in␣
↪quadratic cone
        model.appendcons(name = "risk", ...
                         F = sparse([ zeros(1,n); G_factor_T; diag(sqrt(theta)) ]), ..
↪.
                         g = [gamma; zeros(size(G_factor_T, 1) + n, 1)], ...
                         domain = mosekdomain("qcone", dim = size(G_factor_T, 1) + n␣
↪+ 1 ));

        model.solve(quiet = true);

        % Check if the solution is an optimal point
        % See https://docs.mosek.com/latest/matlabapi/accessing-solution.html about␣
↪handling solution statuses.
        [ok,prosta,solsta] = model.hassolution("interior");
        if ~ ok || solsta ~= "OPTIMAL"
            error("Unexpected solution status");
        else
            x = model.getsolution("interior","x");
            er = mu'*x;
            fprintf("Expected return: %.4e Std. deviation: %.4e\n", er, gamma);
        end
```

9.1.4 Slippage Cost

The basic Markowitz model assumes that there are no costs associated with trading the assets and that the returns of the assets are independent of the amount traded. Neither of those assumptions is usually valid in practice. Therefore, a more realistic model is

$$\begin{array}{lrcl} \text{maximize} & \mu^T x & & \\ \text{subject to} & e^T x + \sum_{j=1}^n T_j(\Delta x_j) & = & w + e^T x^0, \\ & x^T \Sigma x & \leq & \gamma^2, \\ & x & \geq & 0. \end{array} \tag{9.6}$$

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.

9.1.5 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

$$z_j = |x_j - x_j^0| \tag{9.7}$$

but in many cases the constraint may be replaced by the relaxed constraint

$$z_j \geq |x_j - x_j^0|, \tag{9.8}$$

For instance if the universe of assets contains a risk free asset then

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

cannot hold for an optimal solution.

If the optimal solution has the property (9.9) 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 (9.7) and (9.8) are equivalent.

The above observations lead to

$$\begin{array}{lrcll} \text{maximize} & \mu^T x & & & \\ \text{subject to} & e^T x + m^T t & = & w + e^T x^0, & \\ & (\gamma, G^T x) & \in & \mathcal{Q}^{k+1}, & \\ & (t_j, 1, x_j - x_j^0) & \in & \mathcal{P}_3^{2/3,1/3}, & j = 1, \ldots, n, \\ & x & \geq & 0. & \end{array} \tag{9.10}$$

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 9.5 demonstrates how to compute an optimal portfolio when market impact cost are included.

Listing 9.5: Implementation of model (9.10).

```
% A model with variables (x, t)
model = mosekmodel(name = "Impact", ...
                   numvar = n+n, ...
                   objsense = "maximize", ...
                   objective = [mu; zeros(n,1)]);

% Linear constraint
% [ e' m' ]  * [ x; t ]  =   w + e'*x0
model.appendcons(name = "budget", F = [ ones(1,n), m' ], ...
                 domain = mosekdomain("equal", rhs = w + sum(x0)));

% No shortselling, x >= 0 and no other bounds
model.appendcons(F = speye(n, 2*n),  ...
                 domain = mosekdomain("greater than", dim = n, rhs = zeros(n,1)));

% Affine conic constraint representing [ gamma, GT*x ] in quadratic cone
model.appendcons(F = [sparse(1, 2*n); ...
                      GT, sparse(n, n)], ...
                 g = [gamma; zeros(n, 1)], ...
                 domain = mosekdomain("qcone", dim = n + 1));


% Power cone constraints 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

model.appendcons(name = "impact", ...
                 F = sparse(fi, fj, fv, 3*n, 2*n), g = g, ...
                 domain = mosekdomain("power cone", dim = 3, n = n, alpha = [2.0,␣
↪1.0]));

model.solve();

% Check if the solution is an optimal point
% See https://docs.mosek.com/latest/matlabapi/accessing-solution.html about␣
↪handling solution statuses.
[ok,prosta,solsta] = model.hassolution("interior");
if ~ ok || solsta ~= "OPTIMAL"
    error("Unexpected solution status");
else
    xx = model.getsolution("interior","x");
    x = xx(1:n);
end
```

9.1.6 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 (9.6) may be formulated as

$$\begin{array}{lrcll}
\text{maximize} & \mu^T x & & & \\
\text{subject to} & e^T x + f^T y + g^T z & = & w + e^T x^0, & \\
 & (\gamma, G^T x) & \in & \mathcal{Q}^{k+1}, & \\
 & z_j & \geq & x_j - x_j^0, & j = 1, \ldots, n, \\
 & z_j & \geq & x_j^0 - x_j, & j = 1, \ldots, n, \\
 & z_j & \leq & U_j y_j, & j = 1, \ldots, n, \\
 & y_j & \in & \{0, 1\}, & j = 1, \ldots, n, \\
 & x & \geq & 0. &
\end{array} \tag{9.11}$$

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 9.6: Code solving problem (9.11).

```
    % Upper bound on the traded amount
    u = w + sum(x0);

    % A model with variables (x, z, y)
    model = mosekmodel(name = "Impact", ...
                       numvar = n+n+n, ...
                       objsense = "maximize", ...
                       objective = [mu; zeros(n,1); zeros(n,1)], ...
                       intvars = 2*n+1:3*n);          % The variables y are integer

    % y is binary, ie. in [0,1]
    model.appendcons(F = [sparse(n,2*n) speye(n,n)], domain = mosekdomain("gt", dim =␣
↪n, rhs = zeros(n,1)));
    model.appendcons(F = [sparse(n,2*n) speye(n,n)], domain = mosekdomain("lt", dim =␣
↪n, rhs = ones(n,1)));

    % Linear constraints
    % [ e'  g'  f' ]    [ x ]  =   w + e'*x0
    % [ I  -I   0  ] * [ z ]  <=  x0
    % [ I   I   0  ]   [ y ]  >=  x0
    % [ 0   I  -U  ]          <=  0

    % Linear budget constraint
```

(continues on next page)

(continued from previous page)

```
    model.appendcons(name = "budget", F = [ ones(1,n), g', f' ], ...
                     domain = mosekdomain("equal", rhs = u));

    % Linear absolute value model z >= |x-x0|
    model.appendcons(name = "abs1", F = [ speye(n) -speye(n) sparse(n,n) ], ...
                     domain = mosekdomain("lt", dim = n, rhs = x0));
    model.appendcons(name = "abs2", F = [ speye(n) speye(n) sparse(n,n) ], ...
                     domain = mosekdomain("gt", dim = n, rhs = x0));

    % "big-M" constraint z<=Uy
    model.appendcons(name = "bigM", F = [ sparse(n,n) speye(n) -u*speye(n) ], ...
                     domain = mosekdomain("lt", dim = n, rhs = zeros(n,1)));

    % No shortselling, x >= 0
    model.appendcons(F = speye(n, 3*n),  ...
                     domain = mosekdomain("greater than", dim = n, rhs = zeros(n,1)));

    % Affine conic constraint representing [ gamma, GT*x ] in quadratic cone
    model.appendcons(F = [sparse(1, 3*n); ...
                          GT, sparse(n, 2*n)], ...
                     g = [gamma; zeros(n, 1)], ...
                     domain = mosekdomain("qcone", dim = n + 1));

    model.solve();

    % Check if the solution is an optimal point
    % See https://docs.mosek.com/latest/matlabapi/accessing-solution.html about␣
↪handling solution statuses.
    [ok,prosta,solsta] = model.hassolution("integer");
    if ~ ok || solsta ~= "INTEGER_OPTIMAL"
        error("Unexpected solution status");
    else
        xx = model.getsolution("integer","x");
        x = xx(1:n);
        y = xx(2*n+(1:n));
        z = xx(n+(1:n));
    end

    fprintf("\nMarkowitz portfolio optimization with transactions cost")
    fprintf("Expected return: %.4e Std. deviation: %.4e Transactions cost: %.4e", ...
                     mu'*x, gamma, f'*y+g'*z)
```

9.1.7 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:

$$
\begin{array}{lrcll}
\text{maximize} & \mu^T x & & & \\
\text{subject to} & e^T x & = & w + e^T x^0, & \\
 & (\gamma, G^T x) & \in & \mathcal{Q}^{k+1}, & \\
 & z_j & \geq & x_j - x_j^0, & j = 1, \ldots, n, \\
 & z_j & \geq & x_j^0 - x_j, & j = 1, \ldots, n, \\
 & z_j & \leq & U_j y_j, & j = 1, \ldots, n, \\
 & y_j & \in & \{0, 1\}, & j = 1, \ldots, n, \\
 & e^T y & \leq & K, & \\
 & x & \geq & 0, &
\end{array}
\tag{9.12}
$$

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 9.7: Code solving problem (9.12).

```
% Upper bound on the traded amount
u = w + sum(x0);

% A model with variables (x, z, y)
model = mosekmodel(name = "Transaction costs", ...
                   numvar = n+n+n, ...
                   objsense = "maximize", ...
                   objective = [mu; zeros(n,1); zeros(n,1)], ...
                   intvars = 2*n+1:3*n);         % The variables y are integer

% 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

% Linear budget constraint
model.appendcons(name = "budget", F = [ ones(1,n), zeros(1, 2*n) ], ...
                 domain = mosekdomain("equal", rhs = u));

% Linear absolute value model z >= |x-x0|
model.appendcons(name = "abs1", F = [ speye(n) -speye(n) sparse(n,n) ], ...
                 domain = mosekdomain("lt", dim = n, rhs = x0));
model.appendcons(name = "abs2", F = [ speye(n) speye(n) sparse(n,n) ], ...
                 domain = mosekdomain("gt", dim = n, rhs = x0));

% "big-M" constraint z<=Uy
model.appendcons(name = "bigM", F = [ sparse(n,n) speye(n) -u*speye(n) ], ...
                 domain = mosekdomain("lt", dim = n, rhs = zeros(n,1)));

% Cardinality bound sum(y) <= k
model.appendcons(name = "card", F = [ zeros(1, 2*n) ones(1, n) ], ...
                 domain = mosekdomain("lt", rhs = k));

% No shortselling, x >= 0
model.appendcons(F = speye(n, 3*n),  ...
                 domain = mosekdomain("greater than", dim = n, rhs = zeros(n,
```

(continues on next page)

(continued from previous page)

```
↪1)));

        % Affine conic constraint representing [ gamma, GT*x ] in quadratic cone
        model.appendcons(F = [sparse(1, 3*n); ...
                              GT, sparse(n, 2*n)], ...
                         g = [gamma; zeros(n, 1)], ...
                         domain = mosekdomain("qcone", dim = n + 1));

        model.solve(quiet = true);

        % Check if the solution is an optimal point
        % See https://docs.mosek.com/latest/matlabapi/accessing-solution.html about␣
↪handling solution statuses.
        [ok,prosta,solsta] = model.hassolution("integer");
        if ~ ok || solsta ~= "INTEGER_OPTIMAL"
            error("Unexpected solution status");
        else
            xx = model.getsolution("integer","x");
            x = xx(1:n);
            fprintf("Bound: %d   Expected return: %.4e  Solution: [ %s ]\n", k, mu'*x,
↪ sprintf("%.2g ", x));
        end
```

If we solve our running example with $K = 1, \ldots, n$ then we get the following solutions, with increasing expected returns:

```
Bound 1    Solution: 0.0000e+00    0.0000e+00    1.0000e+00    0.0000e+00    0.0000e+00   ␣
↪0.0000e+00    0.0000e+00    0.0000e+00
Bound 2    Solution: 0.0000e+00    0.0000e+00    3.5691e-01    0.0000e+00    0.0000e+00   ␣
↪6.4309e-01    -0.0000e+00   0.0000e+00
Bound 3    Solution: 0.0000e+00    0.0000e+00    1.9258e-01    0.0000e+00    0.0000e+00   ␣
↪5.4592e-01    2.6150e-01    0.0000e+00
Bound 4    Solution: 0.0000e+00    0.0000e+00    2.0391e-01    0.0000e+00    6.7098e-02   ␣
↪4.9181e-01    2.3718e-01    0.0000e+00
Bound 5    Solution: 0.0000e+00    3.1970e-02    1.7028e-01    0.0000e+00    7.0741e-02   ␣
↪4.9551e-01    2.3150e-01    0.0000e+00
Bound 6    Solution: 0.0000e+00    3.1970e-02    1.7028e-01    0.0000e+00    7.0740e-02   ␣
↪4.9551e-01    2.3150e-01    0.0000e+00
Bound 7    Solution: 0.0000e+00    3.1970e-02    1.7028e-01    0.0000e+00    7.0740e-02   ␣
↪4.9551e-01    2.3150e-01    0.0000e+00
Bound 8    Solution: 1.9557e-10    2.6992e-02    1.6706e-01    2.9676e-10    7.1245e-02   ␣
↪4.9559e-01    2.2943e-01    9.6905e-03
```

9.2 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

$$\begin{array}{ll} \text{minimize} & \|Fx - g\|, \\ \text{subject to} & Ax = b, \end{array} \tag{9.13}$$

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 $\|Fx - g\|_2^2$.

9.2.1 Least squares, 2-norm

In the case of the 2-norm we specify the problem directly in conic quadratic form

$$\begin{array}{ll} \text{minimize} & t, \\ \text{subject to} & (t, Fx - g) \in \mathcal{Q}^{k+1}, \\ & Ax = b. \end{array} \tag{9.14}$$

The first constraint of the problem can be represented as an affine conic constraint. This leads to the following model.

Listing 9.8: Script solving problem (9.14)

```
% Least squares regression
% minimize \|Fx-g\|_2
function x = norm_lse(F,g,A,b)
    n = size(F,2);
    k = size(g,1);
    m = size(A,1);

    % A model with variables x(n) and t(1)
    model = mosekmodel(name = "norm_lse", numvar = n + 1, objective = [zeros(n, 1);␣
↪1]);

    % Linear constraints
    model.appendcons(F = [A, zeros(m, 1)], domain = mosekdomain("equal", dim = m, rhs␣
↪= b));

    % Affine conic constraint
    model.appendcons(F = [sparse(1,n), 1; F, sparse(k,1)], g = [0; -g], ...
                     domain = mosekdomain("quadratic cone", dim = k + 1));

    model.solve();
    xx = model.getsolution("any", "x");
    x = xx(1:n);
end
```

9.2.2 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

$$\begin{array}{ll} \text{minimize} & \|Fx - g\|_2 + \gamma \|x\|_2, \\ \text{subject to} & Ax = b, \end{array} \tag{9.15}$$

which can be written explicitly as

$$\begin{array}{ll} \text{minimize} & t_1 + \gamma t_2, \\ \text{subject to} & (t_1, Fx - g) \in \mathcal{Q}^{k+1}, \\ & (t_2, x) \in \mathcal{Q}^{n+1}, \\ & Ax = b. \end{array} \tag{9.16}$$

The implementation is a small extension of that from the previous section.

Listing 9.9: Script solving problem (9.16)

```
% Least squares regression with regularization
% minimize \|Fx-g\|_2 + gamma*\|x\|_2
function x = norm_lse_reg(F,g,A,b,gamma)
    n = size(F,2);
```

(continues on next page)

(continued from previous page)

```
    k = size(g,1);
    m = size(A,1);

    % A model with variables x(n) and t1, t2
    model = mosekmodel(name = "norm_lse_reg", numvar = n + 2, objective = [zeros(n,␣
↪1); 1; gamma]);

    % Linear constraints
    model.appendcons(F = [A, zeros(m, 2)], domain = mosekdomain("equal", dim = m, rhs␣
↪= b));

    % Affine conic constraint for \|Fx-g\|_2
    model.appendcons(F = [sparse(1,n),        1, 0; ...
                          F,              sparse(k,2)], ...
                     g = [0; -g], ...
                     domain = mosekdomain("quadratic cone", dim = k + 1));

    % Affine conic constraint for \|x\|_2
    model.appendcons(F = [sparse(1, n+1),        1; ...
                          speye(n),      sparse(n,2) ], ...
                     domain = mosekdomain("quadratic cone", dim = n + 1));

    model.solve();
    xx = model.getsolution("any", "x");
    x = 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 (9.16). Then they core of the implementation would change as follows:

Listing 9.10: Script solving classical quadratic ridge regression

```
% Least squares regression with regularization
% The "classical" quadratic version
% minimize \|Fx-g\|_2^2 + gamma*\|x\|_2^2
function x = norm_lse_reg_quad(F,g,A,b,gamma)
    n = size(F,2);
    k = size(g,1);
    m = size(A,1);

    % A model with variables x(n) and t1, t2
    model = mosekmodel(name = "norm_lse_reg_quad", numvar = n + 2, objective =␣
↪[zeros(n, 1); 1; gamma]);

    % Linear constraints
    model.appendcons(F = [A, zeros(m, 2)], domain = mosekdomain("equal", dim = m, rhs␣
↪= b));

    % Affine conic constraint for \|Fx-g\|_2^2
    model.appendcons(F = [sparse(1,n) 1 0; ...
                          sparse(1, n+2); ...
                          F sparse(k, 2)], ...
```

(continues on next page)

(continued from previous page)

```
                   g = [0; 0.5; -g], ...
                   domain = mosekdomain("rotated quadratic cone", dim = k + 2));

    % Affine conic constraint for \|x\|_2^2
    model.appendcons(F = [sparse(1, n+1) 1; ...
                          sparse(1, n+2); ...
                          speye(n) sparse(n, 2) ], ...
                   g = [0; 0.5; zeros(n, 1)], ...
                   domain = mosekdomain("rotated quadratic cone", dim = n + 2));

    model.solve();
    xx = model.getsolution("any", "x");
    x = xx(1:n);
end
```

Fig. 9.2 shows the solution to a polynomial fitting problem for a few variants of least squares regression with and without ridge regularization.

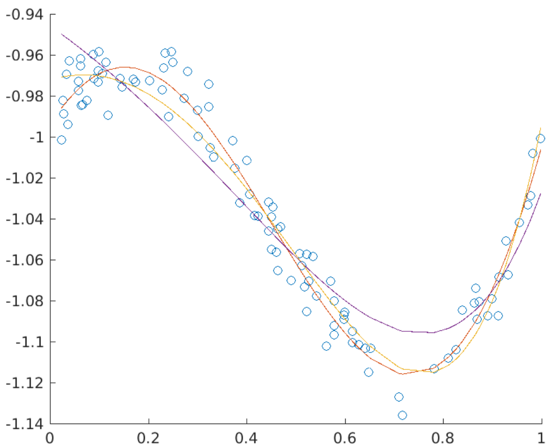

Fig. 9.2: Three fits to a dataset at various levels of regularization.

9.2.3 Lasso regularization

In *lasso* or *least absolute shrinkage and selection operator* the regularization term is the 1-norm of the solution

$$\begin{array}{ll} \text{minimize} & \|Fx-g\|_2 + \gamma\|x\|_1, \\ \text{subject to} & Ax = b. \end{array} \tag{9.17}$$

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 (9.17) 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 9.11: Script solving problem (9.17)

```
% Least squares regression with lasso regularization
% minimize \|Fx-g\|_2 + gamma*\|x\|_1
function x = norm_lse_lasso(F,g,A,b,gamma)
    n = size(F,2);
    k = size(g,1);
    m = size(A,1);

    % A model with variables x(n), u(n) and t1, t2
    model = mosekmodel(name = "norm_lse_lasso", numvar = n + n + 2, objective =␣
↪[zeros(2*n, 1); 1; gamma]);

    % Linear constraints
    model.appendcons(F = [A, zeros(m, 2)], domain = mosekdomain("equal", dim = m, rhs␣
↪= b));

    % u >=. |x|
    model.appendcons(F = [speye(n)  speye(n) sparse(n, 2); ...
                          -speye(n) speye(n) sparse(n, 2)], ...
                     domain = mosekdomain("greater than", dim = 2*n, rhs = zeros(2*n,␣
↪1)));

    % t2 >= sum(u)
    model.appendcons(F = [sparse(1, n), -ones(1, n), 0, 1], ...
                     domain = mosekdomain("greater than", rhs = 0));

    % Affine conic constraint for \|Fx-g\|_2
    model.appendcons(F = [sparse(1, 2*n) 1 0; ...
                          F sparse(k, n + 2)], ...
                     g = [0; -g], ...
                     domain = mosekdomain("quadratic cone", dim = k + 1));

    model.solve();
    xx = model.getsolution("any", "x");
    x = 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
```

9.2.4 p-norm minimization

Now we consider the minimization of the p-norm defined for $p > 1$ as

$$\|y\|_p = \left(\sum_i |y_i|^p\right)^{1/p}. \tag{9.18}$$

We have the optimization problem:

$$\begin{array}{ll} \text{minimize} & \|Fx - g\|_p, \\ \text{subject to} & Ax = b. \end{array} \tag{9.19}$$

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

$$\begin{array}{ll} \text{minimize} & t \\ \text{subject to} & (r_i, t, (Fx-g)_i) \in \mathcal{P}_3^{1/p, 1-1/p}, \\ & e^T r = t, \\ & Ax = b. \end{array} \tag{9.20}$$

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

$$[r_1; t; F_1 x - g_1; \ldots; r_k; t; F_k x - g_k]$$

using an appropriate permutation matrix. This approach is demonstrated in the code below.

Listing 9.12: Script solving problem (9.20)

```
% P-norm minimization
% minimize \|Fx-g\|_p
function x = norm_p_norm(F,g,A,b,p)
    n = size(F,2);
    k = size(g,1);
    m = size(A,1);

    % A model with variables x(n), r(k) and t(1)
    model = mosekmodel(name = "norm_p_norm", numvar = n + k + 1, objective = [zeros(n␣
↪+ k, 1); 1]);

    % Linear constraints
    model.appendcons(F = [A, sparse(m, k + 1)], domain = mosekdomain("equal", dim = m,
↪ rhs = b));

    % t == sum(r)
    model.appendcons(F = [sparse(1, n), ones(1, k), -1], ...
                     domain = mosekdomain("equal", rhs = 0));

    % 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)];
```

(continues on next page)

(continued from previous page)

```
    end

    Fcon = M * [sparse(k, n), speye(k), sparse(k,1); ...
                sparse(k, n+k), ones(k, 1); ...
                F, sparse(k, k+1)];
    gcon = M * [sparse(2*k, 1); -g];

    model.appendcons(F = Fcon, g = gcon, ...
                     domain = mosekdomain("power cone", n = k, dim = 3, alpha = [1; p-
↪1]));

    model.solve();
    xx = model.getsolution("any", "x");
    x = xx(1:n);
end
```

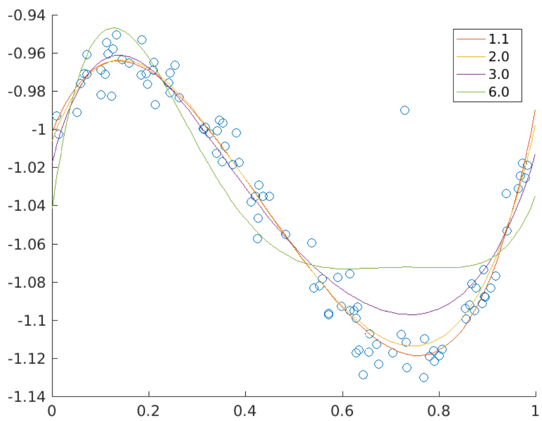

Fig. 9.3: p-norm minimizing fits of a polynomial of degree at most 5 to the data for various values of p.

Chapter 10

Technical guidelines

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

10.1 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.

Constraints can be assigned names when added with *moseknodel.appendcons*, and variable names can be set with *mosekmodel.varname*.

10.2 Timing

Unless otherwise mentioned all parameters, information items and log output entries in **MOSEK** which refer to time measurement are expressed in seconds of wall-clock time.

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.

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. Note also that not all parts of the algorithm can be parallelized, so there are times when CPU utilization is only 1 even if more cores are available.

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*.

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 *mosekmodel.solve*, 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 `"checkinlicense"` command of *mosekenv*.

```
mosekenv("checkinlicense");
```

- The default license path can be changed using the `"licfile"` argument of *mosekmodel.solve*. Note, that although this argument can be used for each solve, the path will in fact be set globally, so the path set at the first optimization will apply to all subsequent ones. To avoid undefined behavior if the **MOSEK** library is temporarily unloaded, we strongly recommend that if the `"licfile"` argument is used at all, then it must be used with exactly the same value for all invocations of *mosekmodel.solve*.

10.5 Deployment

When redistributing a Matlab application using the **MOSEK** API for MATLAB 11.1.1(BETA), the following shared libraries from the **MOSEK** `bin` folder are required:

- Linux : `libmosek64`, `libtbb`,
- Windows : `mosek64`, `tbb`,
- OSX : `libmosek64`, `libtbb`.

together with all the `*.mex*` and `*.m` files from the same folder.

Chapter 11

Problem Formulation and Solutions

In this chapter we will discuss the following topics:

- 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.

11.1 Conic Optimization

Conic optimization is an extension of linear optimizations allowing conic domains to be specified for affine expressions. A conic optimization problem to be solved by **MOSEK** can be written as

$$\begin{array}{lccc} \mbox{minimize} & c^T x + c^f & & \\ \mbox{subject to} & Fx + g & \in & \mathcal{D}, \end{array} \tag{11.1}$$

where

- 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
- $F \in \mathbb{R}^{k \times n}$ is the affine conic constraint matrix.,
- $g \in \mathbb{R}^k$ is the affine conic constraint constant term vector.,
- $\mathcal{D}$ is a Cartesian product of conic domains, namely $\mathcal{D} = \mathcal{D}_1 \times \cdots \times \mathcal{D}_p$, where p is the number of individual affine conic constraints (ACCs), and each domain is one from Sec. 13.8.

The total dimension of the domain $\mathcal{D}$ must be equal to k, the number of rows in F and g.

11.1.1 Duality for Conic Optimization

Corresponding to the primal problem (11.1), there is a dual problem

$$\begin{array}{ll} \text{maximize} & -g^T y + c^f \\ \text{subject to} & F^T y = c, \\ & y \in \mathcal{D}^*, \end{array} \tag{11.2}$$

where

- y are the dual variables for affine conic constraints,
- the dual domain $\mathcal{D}^* = \mathcal{D}_1^* \times \cdots \times \mathcal{D}_p^*$ is a Cartesian product of cones dual to $\mathcal{D}_i$.

One can check that the dual problem of the dual problem is identical to the original primal problem.

A solution y to the dual problem is feasible if it satisfies all the constraints in (11.2). If (11.2) has at least one feasible solution, then (11.2) is *(dual) feasible*, otherwise the problem is *(dual) infeasible*.

A solution

$$(x^*, y^*)$$

is denoted a *primal-dual feasible solution*, if x^* is a solution to the primal problem (11.1) and y^* is a solution to the corresponding dual problem (11.2).

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

$$\begin{array}{l} c^T x^* + c^f - \left(-g^T y^* + c^f\right) \\ = (y^*)^T (F x^* + g) \geq 0. \end{array} \tag{11.3}$$

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*

$$(y^*)^T (F x^* + g) \quad = \quad 0, \tag{11.4}$$

are satisfied.

If (11.1) 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.

11.1.2 Infeasibility for Conic Optimization

Primal Infeasible Problems

If the problem (11.1) 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 system

$$\begin{array}{c} F^T y = 0, \\ g^T y < 0, \\ y \in \mathcal{D}^*. \end{array} \tag{11.5}$$

Such a solution implies that (11.5) is unbounded, and that (11.1) is infeasible.

Dual Infeasible Problems

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

$$\begin{array}{l} c^T x < 0, \\ Fx \in \mathcal{D}. \end{array} \tag{11.6}$$

Such a solution implies that (11.6) is unbounded, and that (11.2) is infeasible.

In case that both the primal problem (11.1) and the dual problem (11.2) are infeasible, **MOSEK** will report only one of the two possible certificates — which one is not defined (**MOSEK** returns the first certificate found).

11.1.3 Minimalization vs. Maximalization

When the objective sense of problem (11.1) is maximization, i.e.

$$\begin{array}{lccc} \text{maximize} & c^T x + c^f & & \\ \text{subject to} & Fx + g & \in & \mathcal{D}, \end{array}$$

the objective sense of the dual problem changes to minimization, and the domain of all dual variables changes sign in comparison to (11.2). The dual problem thus takes the form

$$\begin{array}{ll} \text{minimize} & -g^T y + c^f \\ \text{subject to} & F^T y = c, \\ & -y \in \mathcal{D}^*. \end{array}$$

This means that the duality gap, defined in (11.3) as the primal minus the dual objective value, becomes nonpositive. It follows that the dual objective will always be greater than or equal to the primal objective. The primal infeasibility certificate will be reported by **MOSEK** as a solution to the system

$$\begin{array}{l} F^T y = 0, \\ g^T y > 0, \\ -y \in \mathcal{D}^*. \end{array} \tag{11.7}$$

Similarly, the certificate of dual infeasibility is an x satisfying the requirements of (11.6) such that $c^T x > 0$.

11.2 Linear Optimization

API for MATLAB accepts linear optimization problems of the form

$$\begin{array}{lcccc} \text{minimize} & & c^T x + c^f & & \\ \text{subject to} & & Ax & = & b, \\ b_l & \leq & x & \leq & b_u, \end{array} \tag{11.8}$$

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.
- $b \in \mathbb{R}^m$ is the activity of linear constraints.
- $b_l \in \mathbb{R}^n$ is the lower bound for the variables.

- $b_u \in \mathbb{R}^n$ is the upper bound for the variables.

Lower and upper variable 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 (11.8). If (11.8) has at least one primal feasible solution, then (11.8) is said to be (primal) feasible. In case (11.8) does not have a feasible solution, the problem is said to be *(primal) infeasible*

11.2.1 Duality for Linear Optimization

Corresponding to the primal problem (11.8), there is a dual problem

$$\begin{array}{lrcl} \text{maximize} & b^T y + b_l^T s_l^x - b_u^T s_u^x + c^f & & \\ \text{subject to} & A^T y + s_l^x - s_u^x & = & c, \\ & s_l^x, s_u^x & \geq & 0. \end{array} \tag{11.9}$$

where

- y are the dual variables for constraints,
- s_l^x are the dual variables for lower bounds of variables,
- s_u^x are the dual variables for upper bounds of variables.

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 the corresponding dual variable from the dual problem. For example:

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

A solution

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

to the dual problem is feasible if it satisfies all the constraints in (11.9). If (11.9) has at least one feasible solution, then (11.9) is *(dual) feasible*, otherwise the problem is *(dual) infeasible.*

A solution

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

is denoted a *primal-dual feasible solution*, if (x^*) is a solution to the primal problem (11.8) and $(y^*, (s_l^x)^*, (s_u^x)^*)$ is a solution to the corresponding dual problem (11.9).

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

$$\begin{array}{l} c^T x^* + c^f - \left\{ b^T y^* + b_l^T (s_l^x)^* - b_u^T (s_u^x)^* + c^f \right\} \\ = (A^T y^* + (s_l^x)^* - (s_u^x)^*)^T x - b^T y^* - b_l^T (s_l^x)^* + b_u^T (s_u^x)^* \\ = (y^*)^T (Ax^* - b) + ((s_l^x)^*)^T (x^* - b_l) + ((s_u^x)^*)^T (b_u - x^*) \geq 0 \end{array} \tag{11.10}$$

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*

$$\begin{array}{rcll} y_i^*(Ax - b)_i & = & 0, & i = 0, \ldots, m-1, \\ (s_l^x)_j^*(x_j^* - (b_l)_j) & = & 0, & j = 0, \ldots, n-1, \\ (s_u^x)_j^*((b_u)_j - x_j^*) & = & 0, & j = 0, \ldots, n-1, \end{array}$$

are satisfied.

If (11.8) 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.

11.2.2 Infeasibility for Linear Optimization

Primal Infeasible Problems

If the problem (11.8) 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

$$
\begin{array}{ll}
\text{maximize} & b^T y + b_l^T s_l^x - b_u^T s_u^x \\
\text{subject to} & \\
& A^T y + s_l^x - s_u^x = 0, \\
& s_l^x, s_u^x \geq 0,
\end{array}
\tag{11.11}
$$

such that the objective value is strictly positive, i.e. a solution

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

to (11.11) so that

$$b^T y^* + b_l^T (s_l^x)^* - b_u^T (s_u^x)^* > 0.$$

Such a solution implies that (11.11) is unbounded, and that (11.8) is infeasible.

Dual Infeasible Problems

If the problem (11.9) 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

$$
\begin{array}{lccccc}
\text{minimize} & & & c^T x & & \\
\text{subject to} & & & Ax & = & 0, \\
& \hat{b}_l & \leq & x & \leq & \hat{b}_u,
\end{array}
\tag{11.12}
$$

where

$$\hat{(b_l)}_j := \left\{ \begin{array}{ll} 0 & \text{if } (b_l)_j > -\infty, \\ -\infty & \text{otherwise,} \end{array} \right\} \quad \text{and} \quad \hat{(b_u)}_j := \left\{ \begin{array}{ll} 0 & \text{if } (b_u)_j < \infty, \\ \infty & \text{otherwise,} \end{array} \right\}$$

such that

$$c^T x < 0.$$

Such a solution implies that (11.12) is unbounded, and that (11.9) is infeasible.

In case that both the primal problem (11.8) and the dual problem (11.9) are infeasible, **MOSEK** will report only one of the two possible certificates — which one is not defined (**MOSEK** returns the first certificate found).

11.2.3 Minimalization vs. Maximalization

When the objective sense of problem (11.8) is maximization, i.e.

$$
\begin{array}{lccccc}
\text{maximize} & & & c^T x + c^f & & \\
\text{subject to} & & & Ax & = & b, \\
& b_l & \leq & x & \leq & b_u,
\end{array}
$$

the objective sense of the dual problem changes to minimization, and the domain of all dual variables changes sign in comparison to (11.9). The dual problem thus takes the form

$$
\begin{array}{ll}
\text{minimize} & b^T y + b_l^T s_l^x - b_u^T s_u^x + c^f \\
\text{subject to} & \\
& A^T y + s_l^x - s_u^x = c, \\
& s_l^x, s_u^x \leq 0.
\end{array}
$$

This means that the duality gap, defined in (11.10) 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

$$\begin{aligned} A^T y + s_l^x - s_u^x = 0, \\ s_l^x, s_u^x \leq 0, \end{aligned} \tag{11.13}$$

such that the objective value is strictly negative

$$b^T y^* + b_l^T (s_l^x)^* - b_u^T (s_u^x)^* < 0.$$

Similarly, the certificate of dual infeasibility is an x satisfying the requirements of (11.12) such that $c^T x > 0$.

Below is an outline of the different problem types for quick reference.

Continuous problem formulations

- **Conic optimization (CO)**

  A conic optimization problem (CO) with affine conic constraints has the form:

  $$\begin{array}{lccc} \text{minimize} & c^T x + c^f & & \\ \text{subject to} & Fx + g & \in & \mathcal{D}, \end{array}$$

  where $\mathcal{D}$ is a product of domains from Sec. 13.8.

  This general formulation subsumes also linear optimization (by using linear domains).

- **Linear/simplex optimization (LO)**

  Using the linear/simplex part of the toolbox one can also specify linear problems in the more familiar standard form:

  $$\begin{array}{lccccc} \text{minimize} & & & c^T x + c^f & & \\ \text{subject to} & & & Ax & = & b, \\ & b_l & \leq & x & \leq & b_u. \end{array}$$

Mixed-integer extensions

Coninuous problems can be extended with constraints requiring the mixed-integer optimizer. We outline them briefly here. The continuous part of a mixed-integer problem is formulated according to one of the continuous types above, however only the primal information and solution fields are relevant, there are no dual values and no infeasibility certificates.

- **Integer variables.** Specifies that a subset of variables take integer values, that is

  $$x_I \in \mathbb{Z}$$

  for some index set I.

- **Disjunctive constraints.** Appends disjunctions of the form

  $$\bigvee_{i=1}^{t} \bigwedge_{j=1}^{s_i} (D_{ij} x + d_{ij} \in \mathcal{D}_{ij})$$

  ie. a disjunction of conjunctions of linear constraints, where each $D_{ij} x + d_{ij}$ is an affine expression of the optimization variables and each $\mathcal{D}_{ij}$ is an affine domain. Linear and conic problems can be extended with disjunctive constraints.

Chapter 12

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.

12.1 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 [AGMeszarosX96].

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"`.

In the following we describe in more detail the presolve applied to continuous, i.e., linear and conic optimization problems, see Sec. 12.2 and Sec. 12.3. The mixed-integer optimizer, Sec. 12.4, applies similar techniques. The two most time-consuming steps of the presolve for continuous optimization problems 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.

12.2 Linear Optimization

12.2.1 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.

12.2.2 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

$$\begin{array}{lccl} \text{minimize} & c^T x & & \\ \text{subject to} & Ax & = & b, \\ & x \geq 0. & & \end{array} \tag{12.1}$$

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 (12.1) 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

$$\begin{array}{rcl} Ax - b\tau & = & 0, \\ A^T y + s - c\tau & = & 0, \\ -c^T x + b^T y - \kappa & = & 0, \\ x, s, \tau, \kappa & \geq & 0, \end{array} \tag{12.2}$$

where y and s correspond to the dual variables in (12.1), and τ and κ are two additional scalar variables. Note that the homogeneous model (12.2) 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 (12.2) 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. 11.2 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

$$c^T x^* < 0 \tag{12.3}$$

or

$$b^T y^* > 0 \tag{12.4}$$

is satisfied. If (12.3) is satisfied then x^* is a certificate of dual infeasibility, whereas if (12.4) 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

$$\begin{array}{rcl} \left\| A\frac{x^k}{\tau^k} - b \right\|_\infty & \leq & \epsilon_p (1 + \|b\|_\infty), \\ \left\| A^T \frac{y^k}{\tau^k} + \frac{s^k}{\tau^k} - c \right\|_\infty & \leq & \epsilon_d (1 + \|c\|_\infty), \text{ and} \\ \min\left(\frac{(x^k)^T s^k}{(\tau^k)^2}, \left| \frac{c^T x^k}{\tau^k} - \frac{b^T y^k}{\tau^k} \right| \right) & \leq & \epsilon_g \max\left(1, \frac{\min(\left|c^T x^k\right|, \left|b^T y^k\right|)}{\tau^k} \right), \end{array} \tag{12.5}$$

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 (12.5) is that the optimizer is terminated if

- $\frac{x^k}{\tau^k}$ is approximately primal feasible,
- $\left\{ \frac{y^k}{\tau^k}, \frac{s^k}{\tau^k} \right\}$ is approximately dual feasible, and
- the duality gap is almost zero.

Dual infeasibility certificate

On the other hand, if the trial solution satisfies

$$-\epsilon_i c^T x^k > \frac{\|c\|_\infty}{\max(1, \|b\|_\infty)} \left\| A x^k \right\|_\infty$$

then the problem is declared dual infeasible and x^k is reported as a certificate of dual infeasibility. The motivation for this stopping criterion is as follows: First assume that $\left\| A x^k \right\|_\infty = 0$; then x^k is an exact certificate of dual infeasibility. Next assume that this is not the case, i.e.

$$\left\| A x^k \right\|_\infty > 0,$$

and define

$$\bar{x} := \epsilon_i \frac{\max(1, \|b\|_\infty)}{\|Ax^k\|_\infty \|c\|_\infty} x^k.$$

It is easy to verify that

$$\|A\bar{x}\|_\infty = \epsilon_i \frac{\max(1, \|b\|_\infty)}{\|c\|_\infty} \text{ and } -c^T\bar{x} > 1,$$

which shows $\bar{x}$ is an approximate certificate of dual infeasibility, where ε_i controls the quality of the approximation. A smaller value means a better approximation.

Primal infeasibility certificate

Finally, if

$$\epsilon_i b^T y^k > \frac{\|b\|_\infty}{\max(1, \|c\|_\infty)} \|A^T y^k + s^k\|_\infty$$

then y^k is reported as a certificate of primal infeasibility.

Adjusting optimality criteria

It is possible to adjust the tolerances ε_p, ε_d, ε_g and ε_i using parameters; see table for details.

Table 12.1: 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 (12.5) 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.

The basis identification discussed in Sec. 12.2.2 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
```

(continues on next page)

(continued from previous page)

```
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 indicates if the optimizer chose to solve the primal or dual problem (see *MSK_IPAR_INTPNT_SOLVE_FORM*). The next lines display 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. 12.2.2 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.

12.2.3 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. 12.2.1 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. 11.2 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                    ␣
```

(continues on next page)

(continued from previous page)

```
↪    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).

12.3 Conic Optimization - Interior-point optimizer

For conic optimization problems only an interior-point type optimizer is available.

12.3.1 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:

$$\begin{array}{lccl} \text{minimize} & c^T x & & \\ \text{subject to} & Ax & = & b, \\ & x \in \mathcal{K} & & \end{array} \tag{12.6}$$

where K is a convex cone. The corresponding dual problem is

$$\begin{array}{lccl} \text{maximize} & b^T y & & \\ \text{subject to} & A^T y + s & = & c, \\ & s \in \mathcal{K}^* & & \end{array} \tag{12.7}$$

where $\mathcal{K}^*$ is the dual cone of $\mathcal{K}$. See Sec. 11.1 for definitions.

Since it is not known beforehand whether problem (12.6) 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

$$\begin{array}{rcl} Ax - b\tau & = & 0, \\ A^T y + s - c\tau & = & 0, \\ -c^T x + b^T y - \kappa & = & 0, \\ x & \in & \mathcal{K}, \\ s & \in & \mathcal{K}^*, \\ \tau, \kappa & \geq & 0, \end{array} \tag{12.8}$$

where y and s correspond to the dual variables in (12.6), and τ and κ are two additional scalar variables. Note that the homogeneous model (12.8) 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 (12.8) 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

$$c^T x^* < 0 \tag{12.9}$$

or

$$b^T y^* > 0 \tag{12.10}$$

holds. If (12.9) is satisfied, then x^* is a certificate of dual infeasibility, whereas if (12.10) 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].

12.3.2 Interior-point Termination Criterion

Since computations are performed in finite precision, and for efficiency reasons, it is not possible to solve the homogeneous model exactly in general. Hence, an exact optimal solution or an exact infeasibility certificate cannot be computed and a reasonable termination criterion has to be employed.

In every iteration k of the interior-point algorithm a trial solution

$$(x^k, y^k, s^k, \tau^k, \kappa^k)$$

to the homogeneous model is generated, where

$$x^k \in \mathcal{K}, s^k \in \mathcal{K}^*, \tau^k, \kappa^k > 0.$$

Therefore, it is possible to compute the values:

$$\begin{array}{rcl}
\rho_p^k & = & \arg\min_\rho \left\{ \rho \mid \left\| A\frac{x^k}{\tau^k} - b \right\|_\infty \leq \rho\varepsilon_p (1 + \|b\|_\infty) \right\}, \\
\rho_d^k & = & \arg\min_\rho \left\{ \rho \mid \left\| A^T\frac{y^k}{\tau^k} + \frac{s^k}{\tau^k} - c \right\|_\infty \leq \rho\varepsilon_d (1 + \|c\|_\infty) \right\}, \\
\rho_g^k & = & \arg\min_\rho \left\{ \rho \mid \left(\frac{(x^k)^T s^k}{(\tau^k)^2}, |\frac{c^T x^k}{\tau^k} - \frac{b^T y^k}{\tau^k}| \right) \leq \rho\varepsilon_g \max\left(1, \frac{\min(|c^T x^k|, |b^T y^k|)}{\tau^k}\right) \right\}, \\
\rho_{pi}^k & = & \arg\min_\rho \left\{ \rho \mid \left\| A^T y^k + s^k \right\|_\infty \leq \rho\varepsilon_i b^T y^k,\ b^T y^k > 0 \right\} \text{ and} \\
\rho_{di}^k & = & \arg\min_\rho \left\{ \rho \mid \left\| A x^k \right\|_\infty \leq -\rho\varepsilon_i c^T x^k,\ c^T x^k < 0 \right\}.
\end{array}$$

Note $\varepsilon_p, \varepsilon_d, \varepsilon_g$ and ε_i are nonnegative user specified tolerances.

Optimal Case

Observe ρ_p^k measures how far x^k/τ^k is from being a good approximate primal feasible solution. Indeed if $\rho_p^k \leq 1$, then

$$\left\| A\frac{x^k}{\tau^k} - b \right\|_\infty \leq \varepsilon_p (1 + \|b\|_\infty). \tag{12.11}$$

This shows the violations in the primal equality constraints for the solution x^k/τ^k is small compared to the size of b given ε_p is small.

Similarly, if $\rho_d^k \leq 1$, then $(y^k, s^k)/\tau^k$ is an approximate dual feasible solution. If in addition $\rho_g \leq 1$, then the solution $(x^k, y^k, s^k)/\tau^k$ is approximate optimal because the associated primal and dual objective values are almost identical.

In other words if $\max(\rho_p^k, \rho_d^k, \rho_g^k) \leq 1$, then

$$\frac{(x^k, y^k, s^k)}{\tau^k}$$

is an approximate optimal solution.

Dual Infeasibility Certificate

Next assume that $\rho_{di}^k \leq 1$ and hence

$$\left\| A x^k \right\|_\infty \leq -\varepsilon_i c^T x^k \text{ and } -c^T x^k > 0$$

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.

12.3.3 Adjusting optimality criteria

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 (12.11) 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.

12.3.4 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␣
```

(continues on next page)

(continued from previous page)

```
↪ 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 indicates if the optimizer chose to solve the primal or dual problem (see *MSK_IPAR_INTPNT_SOLVE_FORM*). The next lines display 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. 12.3.1 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`: $\left\|A^T y^k + s^k - c\tau^k\right\|_\infty$. 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 (in seconds).

12.4 The Optimizer for Mixed-Integer Problems

Solving optimization problems where one or more of the variables are constrained to be integer valued is called Mixed-Integer Optimization (MIO). For an introduction to model building with integer variables, the reader is recommended to consult the **MOSEK** Modeling Cookbook, and for further reading we highlight textbooks such as [Wol98] or [CCornuejolsZ14].

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. The mixed-integer optimizer is parallelized, i.e., it can exploit multiple cores during the optimization.

In practice, it often happens that the integer variables in MIO problems are actuall binary variables, taking values in $\{0,1\}$, leading to Mixed- or pure binary problems. In the general setting however, an integer variable may have arbitrary lower and upper bounds.

12.4.1 Branch-and-Bound

In order to succeed in solving mixed-integer problems, it can be useful to have a basic understanding of the underlying solution algorithms. The most important concept in this regard is arguably the so-called Branch-and-Bound algorithm, employed also by **MOSEK**. The more experienced reader may skip this section and advance directly to Sec. 12.4.2.

In order to comprehend Branch-and-Bound, the concept of a *relaxation* is important. Consider for example a mixed-integer linear optimization problem of minimization type

$$\begin{array}{rccll} z^* & = & \text{minimize} & c^T x & \\ & & \text{subject to} & Ax & = & b \\ & & & x \geq 0 & \\ & & & x_j \in \mathbb{Z}, & & \forall j \in \mathcal{J}. \end{array} \tag{12.12}$$

It has the continuous relaxation

$$\begin{array}{rrll} \underline{z} = & \text{minimize} & c^T x & \\ & \text{subject to} & Ax & = b \\ & & x \geq 0, & \end{array} \tag{12.13}$$

simply obtained by ignoring the integrality restrictions. The first step in Branch-and-Bound is to solve this so-called *root* relaxation, which is a continuous optimization problem. Since (12.13) is less constrained than (12.12), one certainly gets

$$\underline{z} \leq z^*,$$

and $\underline{z}$ is therefore called the *objective bound*: it bounds the optimal objective value from below.

After the solution of the root relaxation, in the most likely outcome there will be one or more integer constrained variables with fractional values, i.e., violating the integrality constraints. Branch-and-Bound now takes such a variable, $x_j = f_j \in \mathbb{R} \backslash \mathbb{Z}$ with $j \in \mathcal{J}$, say, and creates two branches leading to relaxations with the additional constraint $x_j \leq \lfloor f_j \rfloor$ or $x_j \geq \lceil f_j \rceil$, respectively. The intuitive idea here is to exclude the undesired fractional value from the outcomes in the two created branches. If the integer variable was actually a binary variable, branching would lead to fixing its value to 0 in one branch, and to 1 in the other.

The Branch-and-Bound process continues in this way and successively solves relaxations and creates branches to refined relaxations. Whenever the solution $\hat{x}$ to some relaxation does not violate any integrality constraints, it is feasible to (12.12) and is called an *integer feasible solution.* There is no guarantee though that it is also optimal, its solution value $\bar{z} := c^T \hat{x}$ is only an upper bound on the optimal objective value,

$$z^* \leq \bar{z}.$$

By the successive addition of constraints in the created branches, the objective bound $\underline{z}$ (now defined as the minimum over all solution values of so far solved relaxations) can only increase during the algorithm. At the same time, the upper bound $\bar{z}$ (the solution value of the best integer feasible solution encountered so far, also called *incumbent solution*) can only decrease during the algorithm. Since at any time we also have

$$\underline{z} \leq z^* \leq \bar{z},$$

objective bound and incumbent solution value are encapsulating the optimal objective value, eventually converging to it.

The Branch-and-Bound scheme can be depicted by means of a tree, where branches and relaxations correspond to edges and nodes. Figure Fig. 12.1 shows an example of such a tree. The strength of Branch-and-Bound is its ability to prune nodes in this tree, meaning that no new child nodes will be created. Pruning can occur in several cases:

- A relaxation leads to an integer feasible solution $\hat{x}$. In this case we may update the incumbent and its solution value $\bar{z}$, but no new branches need to be created.
- A relaxation is infeasible. The subtree rooted at this node cannot contain any feasible relaxation, so it can be discarded.
- A relaxation has a solution value that exceeds $\bar{z}$. The subtree rooted at this node cannot contain any integer feasible solution with a solution value better than the incumbent we already have, so it can be discarded.

Having objective bound and incumbent solution value is a quite fundamental property of Branch-and-Bound, and helps to asses solution quality and control termination of the algorithm, as we detail in the next section. Note that the above explanation is coined for minimization problems, but the Branch-and-bound scheme has a straightforward extension to maximization problems.

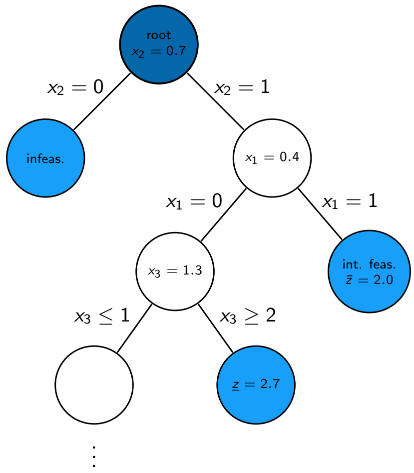

Fig. 12.1: An examplary Branch-and-Bound tree. Pruned nodes are shown in light blue.

12.4.2 Solution quality and termination criteria

The issue of terminating the mixed-integer optimizer is rather delicate. Mixed-integer optimization is generally much harder than continuous optimization; in fact, solving continuous sub-problems is just one component of a mixed-integer optimizer. Despite the ability to prune nodes in the tree, the computational effort required to solve mixed-integer problems grows exponentially with the size of the problem in a worst-case scenario (solving mixed-integer problems is NP-hard). For instance, a problem with n binary variables, may require the solution of 2^n relaxations. The value of 2^n is huge even for moderate values of n. In practice it is often advisable to accept near-optimal or appproximate solutions in order to counteract this complexity burden. The user has numerous possibilities of influencing optimizer termination with various parameters, in particular related to solution quality, and the most important ones are highlighted here.

Solution quality in terms of optimality

In order to assess the quality of any incumbent solution in terms of its objective value, one may check the *optimality gap*, defined as

$$\epsilon = |(\text{incumbent solution value}) - (\text{objective bound})| = |\bar{z} - \underline{z}|.$$

It measures how much the objectives of the incumbent and the optimal solution can deviate in the worst case. Often it is more meaningful to look at the *relative optimality gap*

$$\epsilon_{\text{rel}} = \frac{|\bar{z} - \underline{z}|}{\max(\delta_1, |\bar{z}|)}.$$

This is essentially the above *absolute* optimality gap normalized against the magnitude of the incumbent solution value; the purpose of the (small) constant δ_1 is to avoid overweighing incumbent solution values that are very close to zero. The relative optimality gap can thus be interpreted as answering the question: *"Within what fraction of the optimal solution is the incumbent solution in the worst case?"*

Absolute and relative optimality gaps provide useful means to define termination criteria for the mixed-integer optimizer in **MOSEK**. The idea is to terminate the optimization process as soon as the quality of the incumbent solution, measured in absolute or relative gap, is good enough. In fact, whenever an incumbent solution is located, the criterion

$$\epsilon \leq \delta_2 \text{ or } \epsilon_{\text{rel}} \leq \delta_3$$

is checked. If satisfied, i.e., if either absolute or relative optimality gap are below the thresholds δ_2 or δ_3 (see Table 12.2), the optimizer terminates and reports the incumbent as an optimal solution. The optimality gaps at termination can always be retrieved through the information items *"MSK_DINF_MIO_OBJ_ABS_GAP"* and *"MSK_DINF_MIO_OBJ_REL_GAP"*.

The tolerances discussed above can be adjusted using suitable parameters, see Table 12.2. By default, the optimality parameters δ_2 and δ_3 are quite small, i.e., restrictive. These default values for the absolute and relative gap amount to solving any instance to (almost) optimality: the incumbent is required to be within at most a tiny percentage of the optimal solution. As anticipated, this is not tractable in many practical situations, and one should resort to finding near-optimal solutions quickly rather than insisting on finding the optimal one. It may happen, for example, that an optimal or close-to-optimal solution is found very early by the optimizer, but it spends a huge amount of further computational time for branching, trying to increase $\underline{z}$ that last missing bit: a typical situation that practioneers would want to avoid. The concept of optimality gaps is fundamental for controlling solution quality when resorting to near-optimal solutions.

MIO performance tweaks: termination criteria

One of the first things to do in order to cut down excessive solution time is to increase the relative gap tolerance *MSK_DPAR_MIO_TOL_REL_GAP* to some non-default value, so as to not insist on finding optimal solutions. Typical values could be $0.01, 0.05$ or 0.1, guaranteeing that the delivered solutions lie within $1\%, 5\%$ or 10% of the optimum. Increasing the tolerance will lead to less computational time spent by the optimizer.

Solution quality in terms of feasibility

For an optimizer relying on floating-point arithmetic like the mixed-integer optimizer in **MOSEK**, it may be hard to achieve exact integrality of the solution values of integer variables in most cases, and it makes sense to numerically relax this constraint. Any candidate solution $\hat{x}$ is accepted as integer feasible if the criterion

$$\min(\hat{x_j} - \lfloor \hat{x_j} \rfloor, \lceil \hat{x_j} \rceil - \hat{x_j}) \leq \delta_4 \quad \forall j \in \mathcal{J}$$

is satisfied, meaning that $\hat{x_j}$ is at most δ_4 away from the nearest integer. As above, δ_4 can be adjusted using a parameter, see Table 12.2, and impacts the quality of the acieved solution in terms of integer feasibility. By influencing what solution may be accepted as imcumbent, it can also have an impact on the termination of the optimizer.

MIO performance tweaks: feasibility criteria

Whether increasing the integer feasibility tolerance *MSK_DPAR_MIO_TOL_ABS_RELAX_INT* leads to less solution time is highly problem dependent. Intuitively, the optimizer is more flexible in finding new incumbent soutions so as to improve $\bar{z}$. But this effect has do be examined with care on indivuidual instances: it may worsen solution quality with no effect at all on the solution time. It may in some cases even lead to contrary effects on the solution time.

Table 12.2: Tolerances for the mixed-integer optimizer.

Tolerance	Parameter name	Default value
δ_1	*MSK_DPAR_MIO_REL_GAP_CONST*	1.0e-10
δ_2	*MSK_DPAR_MIO_TOL_ABS_GAP*	0.0
δ_3	*MSK_DPAR_MIO_TOL_REL_GAP*	1.0e-4
δ_4	*MSK_DPAR_MIO_TOL_ABS_RELAX_INT*	1.0e-5

Further controlling optimizer termination

There are more ways to limit the computational effort employed by the mixed-integer optimizer by simply limiting the number of explored branches, solved relaxations or updates of the incumbent solution. When any of the imposed limits is hit, the optimizer terminates and the incumbent solution may be retrieved. See Table 12.3 for a list of corresponding parameters. In contrast to the parameters discussed in Sec. 12.4.2, interfering with these does not maintain any guarantees in terms of solution quality.

Table 12.3: 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.

12.4.3 The Mixed-Integer Log

The Branch-and-Bound scheme from Sec. 12.4.1 is only the basic skeleton of the mixed-integer optimizer in **MOSEK**, and several components are built on top of that in order to enhance its functionality and increase its speed. A mixed-integer optimizer is sometimes referred to as a *"giant bag of tricks"*, and it would be impossible to describe all of these tricks here. Yet, some of the additional components are worth mentioning. They can be influenced by various user parameters, and although the default values of these parameters are optimized to work well on average mixed-integer problems, it may pay off to adjust them for an individual problem, or a specific problem class. The mixed-integer log can give insights on which parameters might be worth an adjustment. Below is a typical log output:

```
Presolve started.
Presolve terminated. Time = 0.23, probing time =  0.09
Presolved problem: 1176 variables, 1344 constraints, 4968 non-zeros
Presolved problem: 328 general integer, 392 binary, 456 continuous
Clique table size: 55
Symmetry factor   : 0.79 (detection time = 0.01)
Removed blocks    : 2
BRANCHES RELAXS   ACT_NDS  DEPTH    BEST_INT_OBJ         BEST_RELAX_OBJ       REL_GAP(
↪%)  TIME
0        0        1        0        8.3888091139e+07     NA                   NA      ␣
↪    0.2
0        1        1        0        8.3888091139e+07     2.5492512136e+07     69.61   ␣
↪    0.3
0        1        1        0        3.1273162420e+07     2.5492512136e+07     18.48   ␣
↪    0.4
0        1        1        0        2.6047699632e+07     2.5492512136e+07     2.13    ␣
↪    0.4
Rooot cut generation started.
0        1        1        0        2.6047699632e+07     2.5492512136e+07     2.13    ␣
↪    0.4
0        2        1        0        2.6047699632e+07     2.5589986247e+07     1.76    ␣
↪    0.4
Rooot cut generation terminated. Time = 0.05
0        4        1        0        2.5990071367e+07     2.5662741991e+07     1.26    ␣
↪    0.5
0        8        1        0        2.5971002767e+07     2.5662741991e+07     1.19    ␣
↪    0.6
0        11       1        0        2.5925040617e+07     2.5662741991e+07     1.01    ␣
↪    0.6
0        12       1        0        2.5915504014e+07     2.5662741991e+07     0.98    ␣
↪    0.6
```

(continues on next page)

(continued from previous page)

```
2        23       1        0        2.5915504014e+07     2.5662741991e+07     0.98    ␣
↪     0.7
14       35       1        0        2.5915504014e+07     2.5662741991e+07     0.98    ␣
↪     0.7

[ ... ]

Objective of best integer solution : 2.578282162804e+07
Best objective bound               : 2.569877601306e+07
Construct solution objective       : Not employed
User objective cut value           : Not employed
Number of cuts generated           : 192
  Number of Gomory cuts            : 52
  Number of CMIR cuts              : 137
  Number of clique cuts            : 3
Number of branches                 : 29252
Number of relaxations solved       : 31280
Number of interior point iterations: 16
Number of simplex iterations       : 105440
Time spend presolving the root     : 0.23
Time spend optimizing the root     : 0.07
Mixed integer optimizer terminated. Time: 6.96
```

The main part here is the iteration log, a progressing series of similar rows reflecting the progress made during the Branch-and-bound process. The columns have the following meanings:

- BRANCHES: Number of branches / nodes generated.
- RELAXS: Number of relaxations solved.
- ACT_NDS: Number of active / non-processed nodes.
- DEPTH: Depth of the last solved node.
- BEST_INT_OBJ: The incumbent solution / best integer objective value, $\bar{z}$.
- BEST_RELAX_OBJ: The objective bound, $\underline{z}$.
- REL_GAP(%): Relative optimality gap, $100\% \cdot \epsilon_{\mathrm{rel}}$
- TIME: Time (in seconds) from the start of optimization.

Also a short solution summary with several statistics is printed. When the solution time for a mixed-integer problem has to be cut down, the log can help to understand where time is spent and what might be improved. We go into some more detail about some further items in the mixed-integer log giving hints about individual components of the optimizer. Alternatively, most of these items can also be retrieved as information items, see Sec. 6.5.

Presolve

Similar to the case of continuous problems, see Sec. 12.1, the mixed-integer optimizer applies various presolve reductions before the actual Branch-and-bound is initiated. The first lines of the mixed-integer log contain a summary of the presolve process, including the time spent therein (`Presolve terminated. Time = 0.23...`). Just as in the continuous case, the use of presolve can be controlled with the parameter *MSK_IPAR_PRESOLVE_USE*. If presolve time seems excessive, instead of switching it off completely one may also try to reduce the time spent in one or more of its individual components. On some models it can also make sense to increase the use of a certain presolve technique. Table Table 12.4 lists some of these with their respective parameters.

Table 12.4: Parameters affecting presolve

Parameter name	Explanation	Possible reference in log
MSK_IPAR_MIO_PROBING_LEVE	Probing aggressivity level.	`... probing time = 0.09`
MSK_IPAR_MIO_SYMMETRY_LEV	Symmetry detection aggressivity level.	`Symmetry factor : 0.79 (detection time = 0.01)`
*MSK_IPAR_MIO_INDEPENDENT_*	Block structure detection level, see Sec. 12.4.3.	`Removed blocks : 2`
*MSK_DPAR_MIO_CLIQUE_TABLE_*	Maximum size of the clqiue table.	`Clique table size: 55`
MSK_IPAR_MIO_PRESOLVE_AGG	Should variable agggregation be enabled?	–

Primal Heuristics

It might happen that the value in the colum `BEST_INT_OBJ` stalls over a long period of log lines, an indication that the optimizer has a hard time improving the incumbent solution, i.e., $\bar{z}$. Solving relaxations in the tree to an integer feasible solution $\hat{x}$ is not the only way to find new incumbent solutions. There is a variety of procedures that, given a mixed-integer problem in a generic form like (12.12), attempt to produce integer feasible solutions in an ad-hoc way. These procedures are called Primal Heuristics, and several of them are implemented in **MOSEK**. For example, whenever a relaxation leads to a fractional solution, one may round the solution values of the integer variables, in various ways, and hope that the outcome is still feasible to the remaining constraints. Primal heuristics are mostly employed while processing the root node, but play a role throughout the whole solution process. The goal of a primal heuristic is to improve the incumbent solution and thus the bound $\bar{z}$, and this can of course affect the quality of the solution that is returned after termination of the optimizer. The user parameters affecting primal heuristics are listed in Table 12.5.

MIO performance tweaks: primal heuristics

- If the mixed-integer optimizer struggles to improve the incumbent solution `BEST_INT_OBJ`, it can be helpful to intensify the use of primal heuristics.
  - Set parameters related to primal heuristics to more aggressive values than the default ones, so that more effort is spent in this component. A List of the respective parameters can be found in Table 12.5. In particular, if the optimizer has difficulties finding any integer feasible solution at all, indicated by `NA` in the column `BEST_INT_OBJ` in the mixed-integer log, one may try to activate a construction heuristic like the Feasibility Pump with *MSK_IPAR_MIO_FEASPUMP_LEVEL* .
  - Specify a good initial solution: In many cases a good feasible solution is either known or easily computed using problem-specific knowledge that the optimizer does not have. If so, it is usually worthwhile to use this as a starting point for the mixed-integer optimizer.
  - For feasibility problems, i.e., problems having a constant objective, the goal is to find a single integer feasible solution, and this can be hard by itself on some instances. Try setting the objective to something meaningful anyway, even if the underlying application does not require this. After all, the feasible set is not changed, but the optimizer might benefit from being able to pursue a concrete goal.
- In rare cases it may also happen that the optimizer spends an excessive amount of time on primal heuristics without drawing any benefit from it, and one may try to limit their use with the respective parameters.

Table 12.5: Parameters affecting primal heuristics

Parameter name	Explanation
MSK_IPAR_MIO_HEURISTIC_LEVEL	Primal heuristics aggressivity level.
MSK_IPAR_MIO_RINS_MAX_NODES	Maximum number of nodes allowed in the RINS heuristic.
MSK_IPAR_MIO_RENS_MAX_NODES	Maximum number of nodes allowed in the RENS heuristic.
MSK_IPAR_MIO_CROSSOVER_MAX_NODES	Maximum number of nodes allowed in the Crossover heuristic.
MSK_IPAR_MIO_OPT_FACE_MAX_NODES	Maximum number of nodes allowed in the optimal face heuristic.
MSK_IPAR_MIO_FEASPUMP_LEVEL	Way of using the Feasibility Pump heuristic.

Cutting Planes

It might as well happen that the value in the colum `BEST_RELAX_OBJ` stalls over a long period of log lines, an indication that the optimizer has a struggles to improve the objective bound $\underline{z}$. A component of the optimizer designed to act on the objective bound is given by Cutting planes, also called cuts or valid inequalities. Cuts do not remove any integer feasible solutions from the feasible set of the mixed-integer problem (12.12). They may, however, remove solutions from the feasible set of the relaxation (12.13), ideally making it a *stronger* relaxation with better objective bound.

As an example, take the constraints

$$2x_1 + 3x_2 + x_3 \le 4, \quad x_1, x_2 \in \{0,1\}, \quad x_3 \ge 0. \tag{12.14}$$

One may realize that there cannot be a feasible solution in which both binary variables take on a value of 1. So certainly

$$x_1 + x_2 \le 1 \tag{12.15}$$

is a valid inequality (there is no integer solution satisfying (12.14), but violating (12.15)). The latter does cut off a portion of the feasible region of the continuous relaxation of (12.14) though, obtained by replacing $x_1, x_2 \in \{0,1\}$ with $x_1, x_2 \in [0,1]$. For example, the fractional point $(x_1, x_2, x_3) = (0.5, 1, 0)$ is feasible to the relaxation, but violates the cut (12.15).

There are many classes of general-purpose cuttting planes that may be generated for a mixed-integer problem in a generic form like (12.12), and **MOSEK**'s mixed-integer optimizer supports several of them. For instance, the above is an example of a so-called clique cut. The most effort on generating cutting planes is spent after the solution of the root relaxation; the beginning and the end of root cut generation is highlighted in the log, and the number of log lines in between reflects to the computational effort spent here. Also the solution summary at the end of the log highlights for each cut class the number of generated cuts. Cuts can also be generated later on in the tree, which is why we also use the term Branch-and-cut, an extension of the basic Branch-and-bound scheme. Cuts aim at improving the objective bound $\underline{z}$ and can thus have significant impact on the solution time. The user parameters affecting cut generation can be seen in Table 12.6.

MIO performance tweaks: cutting planes

- If the mixed-integer optimizer struggles to improve the objective bound `BEST_RELAX_OBJ`, it can be helpful to intensify the use of cutting planes.
  - Some types of cutting planes are not activated by default, but doing so may help to improve the objective bound.
  - The parameters *MSK_DPAR_MIO_TOL_REL_DUAL_BOUND_IMPROVEMENT* and *MSK_IPAR_MIO_CUT_SELECTION_LEVEL* determine how aggressively cuts will be generated and selected.
  - If some valid inequalities can be deduced from problem-specific knowledge that the optimizer does not have, it may be helpful to add these to the problem formulation as constraints. This has to be done with care, since there is a tradeoff between the benefit obtained from an improved objective boud, and the amount of additional constraints that make the relaxations larger.

- In rare cases it may also be observed that the optimizer spends an excessive effort on cutting planes, and one may limit their use with *MSK_IPAR_MIO_MAX_NUM_ROOT_CUT_ROUNDS*, or by disabling a certain type of cutting planes.

Table 12.6: Parameters affecting cutting planes

Parameter name	Explanation
MSK_IPAR_MIO_CUT_CLIQUE	Should clique cuts be enabled?
MSK_IPAR_MIO_CUT_CMIR	Should mixed-integer rounding cuts be enabled?
MSK_IPAR_MIO_CUT_GMI	Should GMI cuts be enabled?
MSK_IPAR_MIO_CUT_IMPLIED_BOUND	Should implied bound cuts be enabled?
MSK_IPAR_MIO_CUT_KNAPSACK_COVER	Should knapsack cover cuts be enabled?
MSK_IPAR_MIO_CUT_LIPRO	Should lift-and-project cuts be enabled?
MSK_IPAR_MIO_CUT_SELECTION_LEVEL	Cut selection aggressivity level.
MSK_IPAR_MIO_MAX_NUM_ROOT_CUT_ROUN	Maximum number of root cut rounds.
MSK_DPAR_MIO_TOL_REL_DUAL_BOUND_IM	Minimum required objective bound improvement during root cut generation.

Restarts

The mixed-integer optimizer employs so-called restarts, i.e., if the progress made while exploring the tree is deemed unsufficient, it might decide to restart the solution process from scratch, possibly making use of the information collected so far. When a restart happens, this is displayed in the log:

```
[ ... ]

1948      4664      699       36        NA                    1.1800000000e+02      NA        ␣
↪     7.2
1970      4693      705       50        NA                    1.1800000000e+02      NA        ␣
↪     7.2

Performed MIP restart 1.
Presolve started.
Presolve terminated. Time = 0.01, probing time =  0.00
Presolved problem: 523 variables, 765 constraints, 3390 non-zeros
Presolved problem: 0 general integer, 404 binary, 119 continuous
Clique table size: 143
BRANCHES RELAXS   ACT_NDS  DEPTH    BEST_INT_OBJ          BEST_RELAX_OBJ        REL_GAP(
↪%)   TIME
1988      4729      1         0         NA                    1.1800000000e+02      NA        ␣
↪     7.3
1988      4730      1         0         4.0000000000e+01      1.1800000000e+02      195.00  ␣
↪     7.3

[ ... ]
```

Restarts tend to be useful especially for hard models. However, in individual cases the optimizer may decide to perform a restart while it would have been better to continue exploring the tree. Their use can be controlled with the parameter *MSK_IPAR_MIO_MAX_NUM_RESTARTS*.

Block decomposition

Sometimes the optimizer faces a model that actually represents two or more completely independent subproblems. For a linear problem such as (12.13), this means that the constraint matrix A is a block-diagonal. Block-diagonal structure can occur after **MOSEK** applies some presolve reductions, e.g., a variable is fixed that was the only variable connecting two otherwise independent subproblems. Or, more rarely, the original model provided by the user is already block-diagonal.

In principle, solving such blocks independently is easier than letting the optimizer work on the single, large model, and **MOSEK** thus tries to exploit this structure. Some blocks may be completely solved and removed from the model during presolve, which can be seen by a line at the end of the presolve summary, see also Sec. 12.4.3. If after presolve there are still independent blocks, **MOSEK** can apply a dedicated algorithm to solve them independently while periodically combining their individual solution statusses (such as incumbent solutions and objective bounds) to the solution status of the original model. Just like the removal of blocks during presolve, the application of this latter strategy is indicated in the log:

```
[ ... ]

15        38        1         0         4.1759800000e+05     3.8354200000e+05     8.16     ␣
↪    0.9
Root cut generation started.
15        38        1         0         4.1759800000e+05     3.8354200000e+05     8.16     ␣
↪    1.1
Root cut generation terminated. Time = 0.11
15        40        1         0         4.1645600000e+05     3.8934425000e+05     6.51     ␣
↪    2.0
15        41        1         0         4.1622400000e+05     3.8934425000e+05     6.46     ␣
↪    2.0
23        52        1         0         4.1622400000e+05     3.8934425000e+05     6.46     ␣
↪    2.0
Decomposition solver started with 5 independent blocks.
532       425       5         118       4.1592600000e+05     3.8935275000e+05     6.39     ␣
↪    4.5
1858      11911     815       286       4.1007800000e+05     3.8946400000e+05     5.03     ␣
↪    11.8

[ ... ]
```

How block-diagonal structure is detected and handled by the optimizer can be controlled with the parameter *MSK_IPAR_MIO_INDEPENDENT_BLOCK_LEVEL* .

12.4.4 Mixed-Integer Nonlinear Optimization

Due to the involved non-linearities, MI(QC)QO or MICO problems are on average harder than MILO problems of comparable size. Yet, the Branch-and-Bound scheme can be applied to these probelm classes in a straightforward manner. The relaxations have to be solved as conic problems with the interior point algorithm in that case, see Sec. 12.3, opposed to MILO where it is often beneficial to solve relaxations with the dual simplex method, see Sec. 12.2.3. There is another solution approach for these types of problems implemented in **MOSEK**, namely the Outer-Approximation algorithm, making use of dynamically refined linear approximations of the non-linearities.

MICO performance tweaks: choice of algorithm

Whether conic Branch-and-Bound or Outer-Approximation is applied to a mixed-integer conic problem can be set with *MSK_IPAR_MIO_CONIC_OUTER_APPROXIMATION* . The best value for this option is highly problem dependent.

12.4.5 Disjunctive constraints

Problems with disjunctive constraints (DJC) are typically reformulated to mixed-integer problems, and even if this is not the case they are solved with an algorithm that is based on the mixed-integer optimizer. In **MOSEK**, these problems thus fall into the realm of MIO. In particular, **MOSEK** automatically attempts to replace any DJC by so called big-M constraints, potentially after transforming it to several, less complicated DJCs. As an example, take the DJC

$$[z = 0] \vee [z = 1, x_1 + x_2 \geq 1000],$$

where $z \in \{0, 1\}$ and $x_1, x_2 \in [0, 750]$. This is an example of a DJC formulation of a so-called indicator constraint. A big-M reformulation is given by

$$x_1 + x_2 \geq 1000 - M \cdot (1 - z),$$

where $M > 0$ is a large constant. The practical difficulty of these constructs is that M should always be sufficiently large, but ideally not larger. Too large values for M can be harmful for the mixed-integer optimizer. During presolve, and taking into account the bounds of the involved variables, **MOSEK** automatically reformulates DJCs to big-M constraints if the required M values do not exceed the parameter *MSK_DPAR_MIO_DJC_MAX_BIGM*. From a performance point-of-view, all DJCs would ideally be linearized to big-Ms after presolve without changing this parameter's default value of 1.0e6. Whether or not this is the case can be seen by retrieving the information item *"MSK_IINF_MIO_PRESOLVED_NUMDJC"*, or by a line in the mixed-integer optimizer's log as in the example below. Both state the number of remaining disjunctions after presolve.

```
Presolved problem: 305 variables, 204 constraints, 708 non-zeros
Presolved problem: 0 general integer, 100 binary, 205 continuous
Presolved problem: 100 disjunctions
Clique table size: 0
BRANCHES RELAXS   ACT_NDS  DEPTH    BEST_INT_OBJ         BEST_RELAX_OBJ       REL_GAP(
↪%)  TIME
0        1        1        0        NA                   0.0000000000e+00     NA      ␣
↪    0.0
0        1        1        0        5.0574653969e+05     0.0000000000e+00     100.00  ␣
↪    0.0

[ ... ]
```

DJC performance tweaks: managing variable bounds

- Always specify the tightest known bounds on the variables of any problem with DJCs, even if they seem trivial from the user-perspective. The mixed-integer optimizer can only benefit from these when reformulating DJCs and thus gain performance; even if bounds don't help with reformulations, it is very unlikely that they hurt the optimizer.
- Increasing *MSK_DPAR_MIO_DJC_MAX_BIGM* can lead to more DJC reformulations and thus increase optimizer speed, but it may in turn hurt numerical solution quality and has to be examined with care. The other way round, on numerically challenging instances with DJCs, decreasing *MSK_DPAR_MIO_DJC_MAX_BIGM* may lead to numerically more robust solutions.

12.4.6 Randomization

A mixed-integer optimizer is usually prone to performance variability, meaning that a small change in either

- problem data, or
- computer hardware, or
- algorithmic parameters

can lead to significant changes in solution time, due to different solution paths in the Branch-and-cut tree. In extreme cases the exact same problem can vary from being solvable in less than a second to seemingly unsolvable in any reasonable amount of time on a different computer.

One practical implication of this is that one should ideally verify whether a seemingly beneficial set of parameters, established experimentally on a single problem, is still beneficial (on average) on a larger set of problems from the same problem class. This protects against making parameter changes that had positive effects only due to random effects on that single problem.

In the absence of a large set of test problems, one may also change the random seed of the optimizer to a series of different values in order to hedge against drawing such wrong conclusions regarding parameters. The random seed, accessible through *MSK_IPAR_MIO_SEED*, impacts for example random tie-breaking in many of the mixed-integer optimizer's components. Changing the random seed can be combined with a permutation of the problem data to further incite randomness, accessible through the parameter *MSK_IPAR_MIO_DATA_PERMUTATION_METHOD*.

Chapter 13

API Reference

- **Command reference:**
  - *Conic Toolbox API* - main interface
  - *Linear/Simplex Toolbox API* - additional simplex interface
  - *Auxiliary functions* - license, environment and other global settings
- **Optimizer parameters:**
  - *Double*, *Integer*, *String*
  - *Full list*
  - *Browse by topic*
- **Optimizer information items:**
  - *Double*, *Integer*, *Long*
- *Optimizer response codes*
- *Constants*
- *List of supported domains*
- *Environment variables*

13.1 Conic Toolbox API

Reference for *moskmodel*, the API for defining and solving conic problems. This is the main component of the API for MATLAB.

For tutorials with explanations and full code examples see Sec. 7, and especially Sec. 7.1.

13.1.1 Construction

`model` = `mosekmodel(name, numvar, objsense, objective, objfixterm, intvars, F, g, domain, solution_x, solution_y, conindexes)`

Creates a model, which represents a conic problem. Most of the problem's data can be set directly in this constructor; problem data can also be input and altered using dedicated methods later. The model object is the user's access point to the problem and solution.

Return

`model` (mosekmodel) – The created model object.

Parameters

- `name` (string) – (optional) Name of the model.
- `numvar` (uint32) – (optional) Number of scalar variables initially in the model.
- `objsense` (string) – (optional) Objective sense: `"min"`, `"minimize"`, `"max"`, `"maximize"`.

- `objective` (double(:,1)) – (optional) Objective vector of length matching `numvar`, if provided.
- `objfixterm` (double) – (optional) Fixed term in the objective.
- `intvars` (uint64(:,1)) – (optional) List of indexes of integer variables.
- `F` (double(:,:)) – (optional) The constraint matrix of affine conic constraints.
- `g` (double(:,1)) – (optional) The constant vector of affine conic constraints. Defaults to 0 if not provided.
- `domain` (mosekdomain(:,1)) – (optional) List of constraint domains, objects created with *mosekdomain*.
- `solution_x` (double(:,1)) – (optional) Initial `x` solution.
- `solution_y` (double(:,1)) – (optional) Initial `y` solution.
- `conindexes` (uint64(:,1)) – (optional) List of row indexes of `F` that define the actual constraints. This allows is to reuse the same `F` row in multiple constraints without repeating the row. Defaults to `[1:size(F,1))]'` if not provided.

Example:

```
model = mosekmodel(objsense = "minimize", ...
                   numvar = 4, ...
                   objective = [ 1 2 5 0 ]', ...
                   F = [ 1 0 1 0; ...
                         0 1 2 -1; ...
                         3 -1 1 0], ...
                   domain = [ mosekdomain("greater than", rhs = 2), ...
                              mosekdomain("less than", rhs = [3 5]', dim = 2) ])
```

13.1.2 Adding data to the model

`conindexes` = `mosekmodel.appendcons(F, g, domain, indexes, name)`

Append a constraint block. A constraint block is a block of the form

$$Fx + g \in D_1 \times \cdots \times D_n$$

Where each D_i is a domain created with *mosekdomain*. The affine expression is in most cases specified by passing `F` and/or `g`.

Return

`conindexes` – Indices of constraint rows.

Parameters

- `F` (double(:,:)) – (optional) Matrix block of the appended constraints.
- `g` (double(:,1)) – (optional) Constant term of the appended constraints. If both `F` and `g` are present then `size(g,1)==size(F,1)` must hold.
- `domain` (mosekdomain(:,1)) – (required) Domain or list of domains.
- `indexes` (uint64(:,1)) – (optional) Indexes of rows already existing in the model. Only allowed if neither `F` nor `g` are specified.
- `name` (string) – (optional) Name of the constraint block.

Example:

```
model.appendcons(name = "sum", ...
                 F = [1 1 1 1], ...
                 domain = mosekdomain("==", rhs = 1.0))
model.appendcons(F = eye(4), g = [0 2 3 1], ...
                 domain = [ mosekdomain("rqcone", dim = 4) ])
```

`mosekmodel.objective(objsense, objective, objfixterm)`

Adds or replaces the objective in the model.

Parameters

- `objsense` (string) – (required) Objective sense: `"min"`, `"minimize"`, `"max"`, `"maximize"`.
- `objective` (double(:,1)) – (required) The objective vector. It's length must match the number of variables in the model.
- `objfixterm` (double(:,1)) – (optional) Fixed term in the objective;

Example:

```
model.objective("minimize", [4 2 0 1]')
%model.objective("maximize", sparse([1], [2], [1.5], 1, 4), objfixterm = 10.0)
```

`mosekmodel.setsolution(part, values)`

Set an initial solution.

Parameters

- `part` (string) – (required) Indicates which part of the solution is requested. Must be either `"x"`, indicating the primal variable values, or `"y"` indicating the dual constraint values.
- `values` (double(:,1)) – (required) The vector of the initial solution.

Example:

```
model.setsolution("x", [1.0 1.0 2.0 3.0]')
```

`varindexes = mosekmodel.appendvars(num, intvars)`

Adds a number of new variables to the model.

Return

`varindexes` (uint32(:,1)) – Indexes of the newly added variables.

Parameters

- `num` (uint32) – (required) Number of scalar variables to add.
- `intvars` (uint32(:,1)) – (optional) List of indexes of integer variables. Index 1 refers to the first new variable.

Example:

```
model.appendvars(2)
model.appendvars(5, intvars = 1:5)
```

`rowindexes = mosekmodel.appendrows(F, g)`

This method is rarely needed. Appends a new block of rows to the problem's affine expression storage. The new rows are initially not used in any constraints. They can later be used to create a conic or disjunctive constraint.

Return

Indexes of newly added rows.

Parameters

- `F` (double(:,:)) – (optional) Matrix block of the appended rows.
- `g` (double(:,1)) – (optional) Constant term of the appended rows. If both `F` and `g` are present then `size(g,1)==size(F,1)` must hold.

13.1.3 Domains

dom = `mosekdomain(which, rhs, dim, n, alpha)`

Defines a conic domain (including linear domains). The domain is a product $D_1 \times \cdots \times D_n$ of `n` identical domains, each of dimension `dim` and type `which`. By default $n = 1$.

Depending on the domain type `which`, some of the arguments `alpha`, `dim` must or must not appear, see the full specification below.

The argument `rhs` is an offset of the domain, ie. a vector b such that instead of D the domain is taken to be $D + b$. For linear domains this corresponds to the right-hand side of the (in-)equality (for instance $a^T x = b$ instead of $a^T x = 0$ for the domain `rzero`).

The possible values of `which` are (for corresponding domain definitions see Sec. 13.8):

- `"rminus"`, `"less than"`, `"lt"`, `"nonpositive"`, `"r-"`, `"<="`: Nonpositive values. Default `dim=1` if not provided.
- `"rplus"`, `"greater than"`, `"gt"`, `"nonnegative"`, `"r+"`, `">="`: Nonnegative values. Default `dim=1` if not provided.
- `"r"`, `"unbounded"`: Unbounded domain. Default `dim=1` if not provided.
- `"zero"`, `"equal"`, `"eq"`, `"fixed"`, `"equals"`, `"=="`: Equal to zero. Default `dim=1` if not provided.
- `"qcone"`, `"quadratic cone"`: Second order cone. `dim` is required and `dim >= 2`.
- `"rqcone"`, `"rotated quadratic cone"`: Rotated second order cone. `dim` is required and `dim >= 3`.
- `"exp"`, `"exponential cone"`, `"dual exp"`: Exponential cone. Dimension is always exactly 3.
- `"dexp"`, `"dual exponential cone"`: Dual exponential cone. Dimension is always exactly 3.
- `"pow"`, `"power cone"`: The power cone. `dim` and `alpha` are required.
- `"dpow"`, `"dual power cone"`, `"dual pow"`: The dual power cone. `dim` and `alpha` are required.
- `"geomean"`, `"geometric mean cone"`: Geometric mean cone. `dim` is required.
- `"dgeomean"`, `"dual geometric mean cone"`, `"dual geomean"`: Dual geometric mean cone. `dim` is required.

Parameters

- `which` (string) – (required) Defines the domain type.
- `rhs` (double(:,1)) – (optional) Domain offset. Shifts the domain by this vector. The leght must equal the total dimension of the domain. See above.
- `dim` (uint64) – (optional) Dimension of the domain. Defaults to 1 for linear domains, to 3 for exponential cones, and required for other domains. See above.
- `n` (uint64) – (optional) Number of times the domain is repeated. Defaults to 1.
- `alpha` (double(:,1)) – (only power cones) The additional vector generating the power cone exponents by normalizing. Its length is the number of variables on the left-hand side of the power cone definiton. If the length is 1, it will be interpreted as `[alpha, 1.0-alpha]`.

Return

`dom` (mosekdomain) – An object representing the defined domain.

Example:

```
dom = mosekdomain("greater than", rhs = 5)                    % >= 5                ␣
↪                                     (1 row in F)
dom = mosekdomain("equals", rhs = ones(10,1), dim = 10)       % >= 1, of␣
↪dimension 10                                 (10 rows in F)
dom = mosekdomain("qcone", dim = 5, n = 10)                   % 10 copies of a 5-
↪dimensional quadratic cone           (50 rows in F)
dom = mosekdomain("exp", n = 10)                              % 10 copies of␣
↪exponential cone                             (30 rows in F)
dom = mosekdomain("pow", alpha = [3 5 2]', dim = 5)           % x1^(3/10)*x2^(5/
↪10)*x3^(2/10) >= sqrt(x4^2 + x5^2)    (5 rows in F)
```

13.1.4 Disjunctions

clause = mosekmodel.clause(F, g, domain, indexes)

A disjunctive constraint consists of a list of clauses. This method creates such a clause, being a conjunction of simple terms of the form $F_i x + g_i \in D_i$, represented as a single affine block and a list of domains created with *mosekdomain*. The affine expressions, if given via F, g, are appended to the affine expression storage. The clauses created with this method can be used in *mosekmodel.appenddisjunction*.

The total dimension of all the domains must equal the number of rows in either F, g or indexes.

Parameters

- F (double(:,:)) – (optional) Matrix block of the clause.
- g (double(:,1)) – (optional) Constant term of the clause. If both F and g are present then size(g,1)==size(F,1) must hold.
- domain (mosekdomain(:,1)) – (required) A domain or an array of domains, each domain defines one simple term in the conjunction.
- indexes (uint64(:,1)) – (optional) Indexes of rows already existing in the model. Only allowed if neither F nor g are specified.

Return

c – Return a clause object representing a constraint block

mosekmodel.appenddisjunction(clauses)

Appends a disjunctive constraint which is a disjunction of the given clauses.

Parameters

clauses (mosekterm(:,1)) – (required) List of clauses.

Example:

```
model.appenddisjunction( [ model.clause(F = [1 -2 0 0 ;
                                              0  0 1 0 ;
                                              0  0 0 1], ...
                                    domain =[ mosekdomain("less than", rhs =␣
↪[-1]), ...
                                              mosekdomain("equal",     dim =␣
↪2, rhs = [0 0]') ]), ...

                          model.clause(F = [0 0 1 -3 ;
                                              1 0 0  0 ;
                                              0 1 0  0], ...
                                    domain = [ mosekdomain("less than", rhs␣
↪= -2), ...
                                               mosekdomain("equal", dim = 2,␣
↪rhs = [0 0]') ]) ], ...
                       name = "disj")
```

13.1.5 Solving and obtaining the solution

mosekmodel.solve(write_to_file, logfile, nosolve, quiet, optserver, analyze, licfile, liccode, param)

Invokes the optimizer to solve the problem.

Parameters

- write_to_file (string) – (optional) Write the problem to this file before solving. The extension specifies the file format.
- logfile (string) – (optional) Write log to this file.
- nosolve (logical) – (optional) Format and load problem, but do not solve.
- quiet (logical) – (optional) Do not print log.

- `optserver` (string) – (optional) Host name for remote optimization.
- `licfile` (string) – (optional) License file path. The same value must be used in all calls in the current process, otherwise the outcome is undefined.
- `liccode` (char(1,:)) – (optional) License key as string. The same value must be used in all calls in the current process, otherwise the outcome is undefined.
- `analyze` (logical) – (optional) Whether to run the problem analyzer after optimization.
- `param` (string(1,:)) – (optional) Array of parameters in the form `[name, value, name, value,...]`.

Example:

```
model.solve()
model.solve(quiet = true)
model.solve(quiet = true, param = ["MSK_DPAR_OPTIMIZER_MAX_TIME", "10.0"])
```

`exists`, `prosta`, `solsta` = `mosekmodel.hassolution(which)`

Check if the specified solution exists and return its statuses.

Return

- `exists` (logical) – Indicates if the solution is available.
- `prosta` (string) – (optional) A string indicating the problem status for the requested solution, one of *`prosta`*. Only returned if `exists` is true.
- `solsta` (string) – (optional) A string indicating the solution status for the requested solution, one of *`solsta`*. Only returned if `exists` is true.

Parameters

`which` (string) – (required) Indicates which solution is requested. Must be one of the values `"interior"`, `"basic"` or `"integer"`.

Example:

```
[exists, prosta, solsta] = model.hassolution("interior")
exists = model.hassolution("integer")
```

`value`, `prosta`, `solsta` = `mosekmodel.getsolution(which, part)`

Return the requested solution values. Throws an error if the requested solution is not available; use *`mosekmodel.hassolution`* to check solution availability.

Return

- `value` (double(:,1)) – The solution values requested.
- `prosta` (string) – A string indicating the problem status for the requested solution, one of *`prosta`*.
- `solsta` (string) – A string indicating the solution status for the requested solution, one of *`solsta`*.

Parameters

- `which` (string) – (required) Indicates which solution is requested. Must be one of the values `"interior"`, `"basic"` or `"integer"` to indicate a specific solution, or `"any"` to indicate any solution available. The latter will choose integer, basic and interior solutions in that order, whichever is first available.
- `part` (string) – (required) Indicates which part of the solution is requested. Must be either `"x"`, indicating the primal variable values, or `"y"` indicating the dual constraint values.

Example:

```
x = model.getsolution("any", "x")
[x, prosta, solsta] = model.getsolution("integer", "x")
```

13.1.6 Auxiliary functions

`mosekmodel.write(filename, param)`

Writes the current problem to a file.

Parameters

- `filename` (string) – (required) File name to write the problem to. The format is determined by the extension.
- `param` (string(1,:)) – (optional) Array of parameters in the form `[name, value, name, value,...]`.

Example:

```
model.write("dump.task.gz")
model.write("save.ptf", param = ["MSK_IPAR_PTF_WRITE_SOLUTIONS", "MSK_ON"])
```

13.1.7 Names

`mosekmodel.varname(indexes, names)`

Adds variable names to the model.

Parameters

- `indexes` (uint64(:,1)) – (required) Indexes of the variables to name.
- `names` (string(:,1)) – (required) Names to assign.

Example:

```
model.varname(1:3, ["b", "tmp1", "tmp2"])
```

13.2 Linear/Simplex Toolbox API

Reference for *moseklinmodel*, the API for defining and solving linear problems. Most linear problems can be solved using the main entry point *mosekmodel*. This part of the API is intended primarily for users who need to use the simplex algorithm and its associated features: basic solution and warm-start.

For tutorials with explanations and full code examples see Sec. 7, and especially Sec. 7.7.

13.2.1 Construction

`model = moseklinmodel(name, numvar, objsense, objective, objfixterm, A, b, blx, bux, solution_x, solution_y, solution_slx, solution_sux, solution_skc, solution_skx, varnames, connames)`

Creates a model, which represents a linear problem. Most of the problem's data can be set directly in this constructor; problem data can also be input and altered using dedicated methods later. The model object is the user's access point to the problem and solution.

Return

`model` (moseklinmodel) – The created linear model object.

Parameters

- `name` (string) – (optional) Name of the model.
- `numvar` (uint32) – (optional) Number of scalar variables initially in the model.
- `objsense` (string) – (optional) Objective sense: `"min"`, `"minimize"`, `"max"`, `"maximize"`.
- `objective` (double(:,1)) – (optional) Objective vector of length matching `numvar`, if provided.
- `objfixterm` (double) – (optional) Fixed term in the objective.
- `A` (double(:,:)) – (optional) The linear constraint matrix.

- b (double(:,1)) – (optional) The linear constraint right-hand side.
- blx (double(:,1)) – (optional) Variable lower bounds.
- bux (double(:,1)) – (optional) Variable upper bounds.
- solution_x (double(:,1)) – (optional) Initial x solution, primal solution values.
- solution_y (double(:,1)) – (optional) Initial y solution, dual values of linear constraints.
- solution_slx (double(:,1)) – (optional) Initial slx solution, dual values of variable lower bounds.
- solution_sux (double(:,1)) – (optional) Initial sux solution, dual values of variable upper bounds.
- solution_skc (string(:,1)) – (optional) Initial skc solution, status keys of constraints.
- solution_skx (string(:,1)) – (optional) Initial skx solution, status keys of variables.
- varnames (string(:,1)) – (optional) Names of variables.
- connames (string(:,1)) – (optional) Names of constraints.

Example:

```
model = moseklinmodel(...
              name = "lo1", ...
              numvar = 6, ...
              objsense = "maximize", ...
              objective = [ 3 1 5 1 0 0 ]', ...
              A = [ 3 1 2 0  0  0 ; ...
                    2 1 3 1 -1  0 ; ...
                    0 2 0 3  0 -1 ], ...
              b = [ 30 15 25 ]', ...
              blx = [ 0.0  0.0 0.0 0.0   0.0 -inf ]', ...
              bux = [ inf 10.0 inf inf   inf  0.0 ]')
```

13.2.2 Adding data to the model

moseklinmodel.appendcons(A, b, x, names)

Append a new constraint block of the form $Ax = b$.

Parameters

- A (double(:,:)) – (required) Matrix block of the appended constraints.
- b (double(:,1)) – (required) Right-hand side of the constraint block. size(b, 1)==size(A,1) must hold.
- x (uint32(:,1)) – (optional) Indexes of variables. If provided, then it is assumed that the constraint blocks only involves the columns with indices in x, and size(A,2)==size(x,1) must hold.
- names (string) – (optional) Names of the constraint rows, size(names, 1)==size(A,1) must hold.

Example:

```
model.appendcons([ 1 1 1 1 1 1 ], ...
                 [10]', ...
                 names = ["sum==10"])
model.appendcons(speye(6), ...
                 ones(6, 1) * 3, ...
                 names = "x_" + (1:6) + "_eq_3")
```

`moseklinmodel.objective(objsense, objective, objfixterm)`

Adds or replaces the objective in the model.

Parameters

- `objsense` (string) – (required) Objective sense: `"min"`, `"minimize"`, `"max"`, `"maximize"`.
- `objective` (double(:,1)) – (required) The objective vector. It's length must match the number of variables in the model.
- `objfixterm` (double(:,1)) – (optional) Fixed term in the objective;

Example:

```
model.objective("minimize", [4 2 0 1 1 0]')
%model.objective("maximize", sparse([1], [2], [1.5], 1, 6), objfixterm = 10.0)
```

`moseklinmodel.setsolution(part, values)`

Set an initial solution.

Parameters

- `part` (string) – (required) Indicates which part of the solution is requested: `"x"` for the primal variable values, `"y"` for dual constraint values, `"slx"`, `"sux"` for duals of variable bounds, `"skc"`, `"skx"` for constraint and variable status keys taken from *`stakey`*.
- `values` (double(:,1)/string(,:1)) – (required) The vector of the initial solution, a double array for numerical values or a string array for status keys taken from *`stakey`*.

Example:

```
model.setsolution("x", [1.0 1.0 2.0 3.0 zeros(1,2)]')
model.setsolution("skx", ["low" "bas" "bas"]')
```

`varindexes = moseklinmodel.appendvars(num, bl, bu, c, A, names)`

Append `num` new variables.

Return

`varindexes` (uint32(:,1)) – Indexes of the newly added variables.

Parameters

- `num` (double) – (required) The number of new variables.
- `bl` (double(:,1)) – (optional) Lower bounds for the new variables.
- `bu` (double(:,1)) – (optional) Upper bounds for the new variables.
- `c` (double(:,1)) – (optional) The objective coefficients for the new variables.
- `A` (double(:,:)) – (optional) Constraint matrix nonzeros for the new columns, appended horizontally to the current constraint matrix.
- `names` (string(1,:)) – (optional) Names of the new variables.

Example:

```
model.appendvars(3, bl = [0, 0, 0], bu = [1, 1, 1], ...
                 names = "tmp_" + (1:3))
model.appendvars(1, bl = [-1], bu = [1], ...
                 A = sparse([2 zeros(1,9)]'), c = [1]', ...
                 names = ["slack"])
```

13.2.3 Solving and obtaining the solution

`moseklinmodel.solve(write_to_file, logfile, nosolve, quiet, optserver, analyze, licfile, liccode, param)`

Invokes the optimizer to solve the problem.

Parameters

- `write_to_file` (string) – (optional) Write the problem to this file before solving. The extension specifies the file format.
- `logfile` (string) – (optional) Write log to this file.
- `nosolve` (logical) – (optional) Format and load problem, but do not solve.
- `quiet` (logical) – (optional) Do not print log.
- `optserver` (string) – (optional) Host name for remote optimization.
- `licfile` (string) – (optional) License file path. The same value must be used in all calls in the current process, otherwise the outcome is undefined.
- `liccode` (char(1,:)) – (optional) License key as string. The same value must be used in all calls in the current process, otherwise the outcome is undefined.
- `analyze` (logical) – (optional) Whether to run the problem analyzer after optimization.
- `param` (string(1,:)) – (optional) Array of parameters in the form `[name, value, name, value,...]`.

Example:

```
model.solve()
model.solve(quiet = true)
model.solve(quiet = true, param = ["MSK_DPAR_OPTIMIZER_MAX_TIME", "10.0"])
```

`exists, prosta, solsta = moseklinmodel.hassolution()`

Check if the basic solution exists and return its statuses.

Return

- `exists` (logical) – Indicates if the basic solution is available.
- `prosta` (string) – (optional) A string indicating the problem status for the basic solution, one of *prosta*. Only returned if `exists` is true.
- `solsta` (string) – (optional) A string indicating the solution status for the basic solution, one of *solsta*. Only returned if `exists` is true.

Example:

```
[exists, prosta, solsta] = model.hassolution()
```

`value, prosta, solsta = moseklinmodel.getsolution(part)`

Return the requested solution values for the basic solution. Throws an error if the basic solution is not available; use *moseklinmodel.hassolution* to check solution availability.

Return

- `value` (double(:,1)/string(:,1)) – The solution values requested. Returns a double array for numerical values and string array for status keys.
- `prosta` (string) – A string indicating the problem status for the basic solution, one of *prosta*.
- `solsta` (string) – A string indicating the solution status for the basic solution, one of *solsta*.

Parameters

`part` (string) – (required) Indicates which part of the solution is requested: `"x"` for the primal variable values, `"y"` for dual constraint values, `"slx"`, `"sux"` for duals of variable bounds, `"skc"`, `"skx"` for constraint and variable status keys taken from *stakey*.

Example:

```
x = model.getsolution("x")
y = model.getsolution("y")
skc = model.getsolution("skc")
```

13.2.4 Modifying the model

`moseklinmodel.setb(b, first)`

Update the constraint right-hand side vector `b` or its slice.

Parameters

- `b` (double(:,1)) – (required) New sequence of values.
- `first` (int32) – (optional) The first index of the slice to be updated. Default equals to 1.

Example:

```
model.setb([3, 4, 5]', first = 2)
```

`moseklinmodel.setvarbounds(bl, bu, first)`

Update the variable bounds for all variables or their slice.

Parameters

- `bl` (double(:,1)) – (optional) New sequence of lower bounds.
- `bu` (double(:,1)) – (optional) New sequence of upper bounds.
- `first` (int32) – (optional) The first index of the slice to be updated. Default equals to 1.

Example:

```
model.setvarbounds(bl = [5], bu = [6], first = 5)
```

`moseklinmodel.setrows(A, first)`

Update a slice of rows in the constraint matrix `A`.

Parameters

- `A` (double(:,:)) – (required) A matrix of new row values. The second dimension `size(A,2)` must equal the number of variables in the problem.
- `first` (int32) – (optional) The first row index of the slice to be updated. Default equals to 1.

Example:

```
model.setrows(zeros(2, model.getnumvar()))
model.setrows(ones(1, model.getnumvar()), first = 3)
```

`moseklinmodel.setcolumns(A, first)`

Update a slice of columns in the constraint matrix `A`.

Parameters

- `A` (double(:,:)) – (required) A matrix of new column values. The first dimension `size(A,1)` must equal the number of constraints in the problem.
- `first` (int32) – (optional) The first column index of the slice to be updated. Default equals to 1.

Example:

```
model.setcolumns(zeros(model.getnumcon(), 3))
model.setcolumns(ones(model.getnumcon(), 1), first = 3)
```

`moseklinmodel.setc(c, first)`

Update the objective vector or its slice.

Parameters

- `c` (double(:,1)) – (required) New sequence of values.
- `first` (int32) – (optional) The first index of the slice to be updated. Default equals to 1.

Example:

```
model.setc(zeros(6, 1), first = 3)
```

13.2.5 Auxiliary functions

`moseklinmodel.write(filename, param)`

Writes the current problem to a file.

Parameters

- `filename` (string) – (required) File name to write the problem to. The format is determined by the extension.
- `param` (string(1,:)) – (optional) Array of parameters in the form `[name, value, name, value,...]`.

Example:

```
model.write("dump.task.gz")
model.write("save.ptf", param = ["MSK_IPAR_PTF_WRITE_SOLUTIONS", "MSK_ON"])
```

13.2.6 Names

`moseklinmodel.varname(indexes, names)`

Adds variable names to the model.

Parameters

- `indexes` (uint32(:,1)) – (required) Indexes of the variables to name.
- `names` (string(:,1)) – (required) Names to assign.

Example:

```
model.varname(2:4, ["b", "tmp1", "tmp2"])
```

`moseklinmodel.conname(indexes, names)`

Adds constraint names to the model.

Parameters

- `indexes` (uint32(:,1)) – (required) Indexes of the constraints to name.
- `names` (string(:,1)) – (required) Names to assign.

Example:

```
model.conname(1:2, ["cons1", "cons2"])
```

13.3 Auxiliary functions

`mosekenv(command, args)`

Additional commands to manage the curent **MOSEK** session. The output type depends on the command.

- `command = "version"`: Return the **MOSEK** version. The output is a triple of integers `[majorver,minorver,revision]`.
- `command = "checkinlicense"`: Checks in all currently held licenses. No output.
- `command = "checkoutlicense"`: Check out a license prior to optimization. The argument `"args"` should be one of the strings `"pts"` or `"pton"` indicating which license part should be checked out. Only relevant for floating licenses. No output.

Parameters

- `command` (string) – (required) The command to execute, possibly with additional arguments. See above.
- `args` (string) – (optional) Additional arguments of the command. See above.

Example:

```
[major,minor,rev] = mosekenv("version")
mosekenv("checkoutlicense", "pton")
mosekenv("checkinlicense")
```

13.4 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*

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_PARAMETERS*
- *MSK_IPAR_PTF_WRITE_SINGLE_PSD_TERMS*
- *MSK_IPAR_PTF_WRITE_SOLUTIONS*
- *MSK_IPAR_PTF_WRITE_TRANSFORM*
- *MSK_IPAR_READ_ASYNC*
- *MSK_IPAR_READ_DEBUG*
- *MSK_IPAR_READ_KEEP_FREE_CON*
- *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_ASYNC*
- *MSK_IPAR_WRITE_BAS_CONSTRAINTS*
- *MSK_IPAR_WRITE_BAS_HEAD*
- *MSK_IPAR_WRITE_BAS_VARIABLES*
- *MSK_IPAR_WRITE_COMPRESSION*
- *MSK_IPAR_WRITE_FREE_CON*
- *MSK_IPAR_WRITE_GENERIC_NAMES*
- *MSK_IPAR_WRITE_IGNORE_INCOMPATIBLE_ITEMS*
- *MSK_IPAR_WRITE_INT_CONSTRAINTS*
- *MSK_IPAR_WRITE_INT_HEAD*
- *MSK_IPAR_WRITE_INT_VARIABLES*
- *MSK_IPAR_WRITE_JSON_INDENTATION*
- *MSK_IPAR_WRITE_LP_FULL_OBJ*
- *MSK_IPAR_WRITE_LP_LINE_WIDTH*
- *MSK_IPAR_WRITE_MPS_FORMAT*
- *MSK_IPAR_WRITE_MPS_INT*
- *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_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_KEY*
- *MSK_SPAR_STAT_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_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_OFF_COL_TRH*
- *MSK_IPAR_INTPNT_ORDER_GP_NUM_SEEDS*
- *MSK_IPAR_INTPNT_ORDER_METHOD*
- *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_HEARTBEAT_SIM_FREQ_TICKS*
- *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_INCLUDE_SUMMARY*
- *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_SENSITIVITY*
- *MSK_IPAR_LOG_SENSITIVITY_OPT*
- *MSK_IPAR_LOG_SIM*
- *MSK_IPAR_LOG_SIM_FREQ*
- *MSK_IPAR_LOG_SIM_FREQ_GIGA_TICKS*
- *MSK_IPAR_LOG_STORAGE*

Mixed-integer optimization

- *MSK_DPAR_MIO_CLIQUE_TABLE_SIZE_FACTOR*
- *MSK_DPAR_MIO_DJC_MAX_BIGM*
- *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_CONFLICT_ANALYSIS_LEVEL*
- *MSK_IPAR_MIO_CONIC_OUTER_APPROXIMATION*
- *MSK_IPAR_MIO_CONSTRUCT_SOL*
- *MSK_IPAR_MIO_CROSSOVER_MAX_NODES*
- *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_LIPRO*
- *MSK_IPAR_MIO_CUT_SELECTION_LEVEL*
- *MSK_IPAR_MIO_DATA_PERMUTATION_METHOD*
- *MSK_IPAR_MIO_DUAL_RAY_ANALYSIS_LEVEL*
- *MSK_IPAR_MIO_FEASPUMP_LEVEL*
- *MSK_IPAR_MIO_HEURISTIC_LEVEL*
- *MSK_IPAR_MIO_INDEPENDENT_BLOCK_LEVEL*
- *MSK_IPAR_MIO_MAX_NUM_BRANCHES*
- *MSK_IPAR_MIO_MAX_NUM_RELAXS*
- *MSK_IPAR_MIO_MAX_NUM_RESTARTS*
- *MSK_IPAR_MIO_MAX_NUM_ROOT_CUT_ROUNDS*
- *MSK_IPAR_MIO_MAX_NUM_SOLUTIONS*
- *MSK_IPAR_MIO_MEMORY_EMPHASIS_LEVEL*
- *MSK_IPAR_MIO_MIN_REL*

- *MSK_IPAR_MIO_NODE_OPTIMIZER*
- *MSK_IPAR_MIO_NODE_SELECTION*
- *MSK_IPAR_MIO_NUMERICAL_EMPHASIS_LEVEL*
- *MSK_IPAR_MIO_OPT_FACE_MAX_NODES*
- *MSK_IPAR_MIO_PERSPECTIVE_REFORMULATE*
- *MSK_IPAR_MIO_PROBING_LEVEL*
- *MSK_IPAR_MIO_PROPAGATE_OBJECTIVE_CONSTRAINT*
- *MSK_IPAR_MIO_QCQO_REFORMULATION_METHOD*
- *MSK_IPAR_MIO_RENS_MAX_NODES*
- *MSK_IPAR_MIO_RINS_MAX_NODES*
- *MSK_IPAR_MIO_ROOT_OPTIMIZER*
- *MSK_IPAR_MIO_SEED*
- *MSK_IPAR_MIO_SYMMETRY_LEVEL*
- *MSK_IPAR_MIO_VAR_SELECTION*
- *MSK_IPAR_MIO_VB_DETECTION_LEVEL*

Output information

- *MSK_IPAR_HEARTBEAT_SIM_FREQ_TICKS*
- *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_INCLUDE_SUMMARY*
- *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_SENSITIVITY*

- *MSK_IPAR_LOG_SENSITIVITY_OPT*
- *MSK_IPAR_LOG_SIM*
- *MSK_IPAR_LOG_SIM_FREQ*
- *MSK_IPAR_LOG_SIM_FREQ_GIGA_TICKS*
- *MSK_IPAR_LOG_STORAGE*
- *MSK_IPAR_MAX_NUM_WARNINGS*

Overall solver

- *MSK_IPAR_BI_CLEAN_OPTIMIZER*
- *MSK_IPAR_LICENSE_WAIT*
- *MSK_IPAR_MIO_MODE*
- *MSK_IPAR_OPTIMIZER*
- *MSK_IPAR_PRESOLVE_MAX_NUM_REDUCTIONS*
- *MSK_IPAR_PRESOLVE_USE*
- *MSK_IPAR_PRIMAL_REPAIR_OPTIMIZER*
- *MSK_IPAR_SENSITIVITY_ALL*
- *MSK_IPAR_SENSITIVITY_TYPE*
- *MSK_IPAR_SIM_PRECISION*

Overall system

- *MSK_IPAR_AUTO_UPDATE_SOL_INFO*
- *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_OPTSERVER_HOST*
- *MSK_SPAR_REMOTE_TLS_CERT*
- *MSK_SPAR_REMOTE_TLS_CERT_PATH*

Presolve

- *MSK_DPAR_FOLDING_TOL_EQ*
- *MSK_DPAR_PRESOLVE_TOL_ABS_LINDEP*
- *MSK_DPAR_PRESOLVE_TOL_PRIMAL_INFEAS_PERTURBATION*
- *MSK_DPAR_PRESOLVE_TOL_REL_LINDEP*
- *MSK_DPAR_PRESOLVE_TOL_S*
- *MSK_DPAR_PRESOLVE_TOL_X*
- *MSK_IPAR_FOLDING_USE*
- *MSK_IPAR_MIO_PRESOLVE_AGGREGATOR_USE*
- *MSK_IPAR_PRESOLVE_ELIMINATOR_MAX_FILL*
- *MSK_IPAR_PRESOLVE_ELIMINATOR_MAX_NUM_TRIES*
- *MSK_IPAR_PRESOLVE_LINDEP_ABS_WORK_TRH*
- *MSK_IPAR_PRESOLVE_LINDEP_NEW*
- *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*

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_SIM_PRECISION_SCALING_EXTENDED*
- *MSK_DPAR_SIM_PRECISION_SCALING_NORMAL*
- *MSK_DPAR_SIMPLEX_ABS_TOL_PIV*
- *MSK_IPAR_BASIS_SOLVE_USE_PLUS_ONE*
- *MSK_IPAR_HEARTBEAT_SIM_FREQ_TICKS*
- *MSK_IPAR_LOG_SIM*
- *MSK_IPAR_LOG_SIM_FREQ*

- *MSK_IPAR_LOG_SIM_FREQ_GIGA_TICKS*
- *MSK_IPAR_SIM_BASIS_FACTOR_USE*
- *MSK_IPAR_SIM_DEGEN*
- *MSK_IPAR_SIM_DETECT_PWL*
- *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_PRECISION_BOOST*
- *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_SWITCH_OPTIMIZER*

Solution input/output

- *MSK_IPAR_INFEAS_REPORT_AUTO*
- *MSK_IPAR_SOL_FILTER_KEEP_BASIC*
- *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_TICKS*
- *MSK_DPAR_OPTIMIZER_MAX_TIME*
- *MSK_DPAR_SIM_PRECISION_SCALING_EXTENDED*
- *MSK_DPAR_SIM_PRECISION_SCALING_NORMAL*
- *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*
- *MSK_IPAR_GETDUAL_CONVERT_LMIS*
- *MSK_IPAR_NG*
- *MSK_IPAR_REMOTE_USE_COMPRESSION*

13.5 Parameters (alphabetical list sorted by type)

- *Double parameters*
- *Integer parameters*
- *String parameters*

13.5.1 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]

Example
`param = [ "MSK_DPAR_ANA_SOL_INFEAS_TOL", "1e-6" ]`

Groups
Analysis

`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]

Example
`param = [ "MSK_DPAR_BASIS_REL_TOL_S", "1.0e-12" ]`

Groups
Simplex optimizer, *Termination criteria*

`MSK_DPAR_BASIS_TOL_S`

Maximum absolute dual bound violation in an optimal basic solution.

Default
1.0e-6

Accepted
[1.0e-9; +inf]

Example
`param = [ "MSK_DPAR_BASIS_TOL_S", "1.0e-6" ]`

Groups
Simplex optimizer, *Termination criteria*

`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]

Example
`param = [ "MSK_DPAR_BASIS_TOL_X", "1.0e-6" ]`

Groups
Simplex optimizer, *Termination criteria*

`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]

Example
`param = [ "MSK_DPAR_DATA_SYM_MAT_TOL", "1.0e-12" ]`

Groups
Data check

`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]

Example

`param = [ "MSK_DPAR_DATA_SYM_MAT_TOL_HUGE", "1.0e20" ]`

Groups

Data check

`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]

Example

`param = [ "MSK_DPAR_DATA_SYM_MAT_TOL_LARGE", "1.0e10" ]`

Groups

Data check

`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]

Example

`param = [ "MSK_DPAR_DATA_TOL_AIJ_HUGE", "1.0e20" ]`

Groups

Data check

`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]

Example

`param = [ "MSK_DPAR_DATA_TOL_AIJ_LARGE", "1.0e10" ]`

Groups

Data check

`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]

Example

`param = [ "MSK_DPAR_DATA_TOL_BOUND_INF", "1.0e16" ]`

Groups

Data check

`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]

Example
`param = [ "MSK_DPAR_DATA_TOL_BOUND_WRN", "1.0e8" ]`

Groups
Data check

`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]

Example
`param = [ "MSK_DPAR_DATA_TOL_C_HUGE", "1.0e16" ]`

Groups
Data check

`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]

Example
`param = [ "MSK_DPAR_DATA_TOL_CJ_LARGE", "1.0e8" ]`

Groups
Data check

`MSK_DPAR_DATA_TOL_QIJ`

Absolute zero tolerance for elements in Q matrices.

Default
1.0e-16

Accepted
[0.0; +inf]

Example
`param = [ "MSK_DPAR_DATA_TOL_QIJ", "1.0e-16" ]`

Groups
Data check

`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]

Example

param = ["MSK_DPAR_DATA_TOL_X", "1.0e-8"]

Groups

Data check

MSK_DPAR_FOLDING_TOL_EQ

Tolerance for coefficient equality during folding.

Default

1e-9

Accepted

[0.0; +inf]

Example

param = ["MSK_DPAR_FOLDING_TOL_EQ", "1e-9"]

Groups

Presolve

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]

Example

param = ["MSK_DPAR_INTPNT_CO_TOL_DFEAS", "1.0e-8"]

See also

MSK_DPAR_INTPNT_CO_TOL_NEAR_REL

Groups

Interior-point method, *Termination criteria*, *Conic interior-point method*

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]

Example

param = ["MSK_DPAR_INTPNT_CO_TOL_INFEAS", "1.0e-12"]

Groups

Interior-point method, *Termination criteria*, *Conic interior-point method*

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]

Example

param = ["MSK_DPAR_INTPNT_CO_TOL_MU_RED", "1.0e-8"]

Groups

Interior-point method, *Termination criteria*, *Conic interior-point method*

`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]

Example
`param = [ "MSK_DPAR_INTPNT_CO_TOL_NEAR_REL", "1000" ]`

Groups
Interior-point method, *Termination criteria*, *Conic interior-point method*

`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]

Example
`param = [ "MSK_DPAR_INTPNT_CO_TOL_PFEAS", "1.0e-8" ]`

See also
MSK_DPAR_INTPNT_CO_TOL_NEAR_REL

Groups
Interior-point method, *Termination criteria*, *Conic interior-point method*

`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]

Example
`param = [ "MSK_DPAR_INTPNT_CO_TOL_REL_GAP", "1.0e-8" ]`

See also
MSK_DPAR_INTPNT_CO_TOL_NEAR_REL

Groups
Interior-point method, *Termination criteria*, *Conic interior-point method*

`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]

Example
`param = [ "MSK_DPAR_INTPNT_QO_TOL_DFEAS", "1.0e-8" ]`

See also
MSK_DPAR_INTPNT_QO_TOL_NEAR_REL

Groups
Interior-point method, *Termination criteria*

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]

Example
`param = [ "MSK_DPAR_INTPNT_QO_TOL_INFEAS", "1.0e-12" ]`

Groups
Interior-point method, *Termination criteria*

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]

Example
`param = [ "MSK_DPAR_INTPNT_QO_TOL_MU_RED", "1.0e-8" ]`

Groups
Interior-point method, *Termination criteria*

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]

Example
`param = [ "MSK_DPAR_INTPNT_QO_TOL_NEAR_REL", "1000" ]`

Groups
Interior-point method, *Termination criteria*

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]

Example
`param = [ "MSK_DPAR_INTPNT_QO_TOL_PFEAS", "1.0e-8" ]`

See also
MSK_DPAR_INTPNT_QO_TOL_NEAR_REL

Groups
Interior-point method, *Termination criteria*

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]

Example
```
param = [ "MSK_DPAR_INTPNT_QO_TOL_REL_GAP", "1.0e-8" ]
```

See also
MSK_DPAR_INTPNT_QO_TOL_NEAR_REL

Groups
Interior-point method, *Termination criteria*

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]

Example
```
param = [ "MSK_DPAR_INTPNT_TOL_DFEAS", "1.0e-8" ]
```

Groups
Interior-point method, *Termination criteria*

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]

Example
```
param = [ "MSK_DPAR_INTPNT_TOL_DSAFE", "1.0" ]
```

Groups
Interior-point method

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]

Example
```
param = [ "MSK_DPAR_INTPNT_TOL_INFEAS", "1.0e-10" ]
```

Groups
Interior-point method, *Termination criteria*

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]

Example
: `param = [ "MSK_DPAR_INTPNT_TOL_MU_RED", "1.0e-16" ]`

Groups
: *Interior-point method*, *Termination criteria*

`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]

Example
: `param = [ "MSK_DPAR_INTPNT_TOL_PATH", "1.0e-8" ]`

Groups
: *Interior-point method*

`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]

Example
: `param = [ "MSK_DPAR_INTPNT_TOL_PFEAS", "1.0e-8" ]`

Groups
: *Interior-point method*, *Termination criteria*

`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]

Example
: `param = [ "MSK_DPAR_INTPNT_TOL_PSAFE", "1.0" ]`

Groups
: *Interior-point method*

`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]

Example
: `param = [ "MSK_DPAR_INTPNT_TOL_REL_GAP", "1.0e-8" ]`

Groups
: *Termination criteria*, *Interior-point method*

`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]

Example
`param = [ "MSK_DPAR_INTPNT_TOL_REL_STEP", "0.9999" ]`

Groups
Interior-point method

`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]

Example
`param = [ "MSK_DPAR_INTPNT_TOL_STEP_SIZE", "1.0e-6" ]`

Groups
Interior-point method

`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
-INFINITY

Accepted
[-inf; +inf]

Example
`param = [ "MSK_DPAR_LOWER_OBJ_CUT", "-INFINITY" ]`

See also
MSK_DPAR_LOWER_OBJ_CUT_FINITE_TRH

Groups
Termination criteria

`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]

Example
`param = [ "MSK_DPAR_LOWER_OBJ_CUT_FINITE_TRH", "-0.5e30" ]`

Groups
Termination criteria

MSK_DPAR_MIO_CLIQUE_TABLE_SIZE_FACTOR

Controlls the maximum size of the clique table as a factor of the number of nonzeros in the A matrix. A negative value implies **MOSEK** decides.

Default
-1

Accepted
[-1; +inf]

Example
`param = [ "MSK_DPAR_MIO_CLIQUE_TABLE_SIZE_FACTOR", "-1" ]`

Groups
Mixed-integer optimization

MSK_DPAR_MIO_DJC_MAX_BIGM

Maximum allowed big-M value when reformulating disjunctive constraints to linear constraints. Higher values make it more likely that a disjunction is reformulated to linear constraints, but also increase the risk of numerical problems.

Default
1.0e6

Accepted
[0; +inf]

Example
`param = [ "MSK_DPAR_MIO_DJC_MAX_BIGM", "1.0e6" ]`

Groups
Mixed-integer optimization

MSK_DPAR_MIO_MAX_TIME

This parameter limits the maximum time spent by the mixed-integer optimizer (in seconds). A negative number means infinity.

Default
-1.0

Accepted
[-inf; +inf]

Example
`param = [ "MSK_DPAR_MIO_MAX_TIME", "-1.0" ]`

Groups
Mixed-integer optimization, *Termination criteria*

MSK_DPAR_MIO_REL_GAP_CONST

This value is used to compute the relative gap for the solution to a mixed-integer optimization problem.

Default
1.0e-10

Accepted
[1.0e-15; +inf]

Example
`param = [ "MSK_DPAR_MIO_REL_GAP_CONST", "1.0e-10" ]`

Groups
Mixed-integer optimization, *Termination criteria*

MSK_DPAR_MIO_TOL_ABS_GAP

Absolute optimality tolerance employed by the mixed-integer optimizer.

Default
0.0

Accepted
[0.0; +inf]

Example
```
param = [ "MSK_DPAR_MIO_TOL_ABS_GAP", "0.0" ]
```

Groups
Mixed-integer optimization

`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]

Example
```
param = [ "MSK_DPAR_MIO_TOL_ABS_RELAX_INT", "1.0e-5" ]
```

Groups
Mixed-integer optimization

`MSK_DPAR_MIO_TOL_FEAS`

Feasibility tolerance for mixed integer solver.

Default
1.0e-6

Accepted
[1e-9; 1e-3]

Example
```
param = [ "MSK_DPAR_MIO_TOL_FEAS", "1.0e-6" ]
```

Groups
Mixed-integer optimization

`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]

Example
```
param = [ "MSK_DPAR_MIO_TOL_REL_DUAL_BOUND_IMPROVEMENT", "0.0" ]
```

Groups
Mixed-integer optimization

`MSK_DPAR_MIO_TOL_REL_GAP`

Relative optimality tolerance employed by the mixed-integer optimizer.

Default
1.0e-4

Accepted
[0.0; +inf]

Example
```
param = [ "MSK_DPAR_MIO_TOL_REL_GAP", "1.0e-4" ]
```

Groups
Mixed-integer optimization, *Termination criteria*

`MSK_DPAR_OPTIMIZER_MAX_TICKS`

CURRENTLY NOT IN USE.

Maximum amount of ticks the optimizer is allowed to spent on the optimization. A negative number means infinity.

Default
-1.0

Accepted
[-inf; +inf]

Example
`param = [ "MSK_DPAR_OPTIMIZER_MAX_TICKS", "-1.0" ]`

Groups
Termination criteria

`MSK_DPAR_OPTIMIZER_MAX_TIME`

Maximum amount of time the optimizer is allowed to spent on the optimization (in seconds). A negative number means infinity.

Default
-1.0

Accepted
[-inf; +inf]

Example
`param = [ "MSK_DPAR_OPTIMIZER_MAX_TIME", "-1.0" ]`

Groups
Termination criteria

`MSK_DPAR_PRESOLVE_TOL_ABS_LINDEP`

Absolute tolerance employed by the linear dependency checker.

Default
1.0e-6

Accepted
[0.0; +inf]

Example
`param = [ "MSK_DPAR_PRESOLVE_TOL_ABS_LINDEP", "1.0e-6" ]`

Groups
Presolve

`MSK_DPAR_PRESOLVE_TOL_PRIMAL_INFEAS_PERTURBATION`

The presolve is allowed to perturb a bound on a constraint or variable by this amount if it removes an infeasibility.

Default
1.0e-6

Accepted
[0.0; +inf]

Example
```
param = [ "MSK_DPAR_PRESOLVE_TOL_PRIMAL_INFEAS_PERTURBATION", "1.0e-6
" ]
```

Groups
Presolve

`MSK_DPAR_PRESOLVE_TOL_REL_LINDEP`

Relative tolerance employed by the linear dependency checker.

Default
1.0e-10

Accepted
: [0.0; +inf]

Example
: `param = [ "MSK_DPAR_PRESOLVE_TOL_REL_LINDEP", "1.0e-10" ]`

Groups
: *Presolve*

`MSK_DPAR_PRESOLVE_TOL_S`

Absolute zero tolerance employed for s_i in the presolve.

Default
: 1.0e-8

Accepted
: [0.0; +inf]

Example
: `param = [ "MSK_DPAR_PRESOLVE_TOL_S", "1.0e-8" ]`

Groups
: *Presolve*

`MSK_DPAR_PRESOLVE_TOL_X`

Absolute zero tolerance employed for x_j in the presolve.

Default
: 1.0e-8

Accepted
: [0.0; +inf]

Example
: `param = [ "MSK_DPAR_PRESOLVE_TOL_X", "1.0e-8" ]`

Groups
: *Presolve*

`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]

Example
: `param = [ "MSK_DPAR_QCQO_REFORMULATE_REL_DROP_TOL", "1e-15" ]`

Groups
: *Interior-point method*

`MSK_DPAR_SEMIDEFINITE_TOL_APPROX`

Tolerance to define a matrix to be positive semidefinite.

Default
: 1.0e-10

Accepted
: [1.0e-15; +inf]

Example
: `param = [ "MSK_DPAR_SEMIDEFINITE_TOL_APPROX", "1.0e-10" ]`

Groups
: *Data check*

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]

Example
`param = [ "MSK_DPAR_SIM_LU_TOL_REL_PIV", "0.01" ]`

Groups
Basis identification, *Simplex optimizer*

MSK_DPAR_SIM_PRECISION_SCALING_EXTENDED

Experimental. Usage not recommended.

Default
2.0

Accepted
[1.0; +inf]

Example
`param = [ "MSK_DPAR_SIM_PRECISION_SCALING_EXTENDED", "2.0" ]`

Groups
Simplex optimizer, *Termination criteria*

MSK_DPAR_SIM_PRECISION_SCALING_NORMAL

Experimental. Usage not recommended.

Default
1.0

Accepted
[1.0; +inf]

Example
`param = [ "MSK_DPAR_SIM_PRECISION_SCALING_NORMAL", "1.0" ]`

Groups
Simplex optimizer, *Termination criteria*

MSK_DPAR_SIMPLEX_ABS_TOL_PIV

Absolute pivot tolerance employed by the simplex optimizers.

Default
1.0e-7

Accepted
[1.0e-12; +inf]

Example
`param = [ "MSK_DPAR_SIMPLEX_ABS_TOL_PIV", "1.0e-7" ]`

Groups
Simplex optimizer

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
INFINITY

Accepted
[-inf; +inf]

Example
`param = [ "MSK_DPAR_UPPER_OBJ_CUT", "INFINITY" ]`

See also
MSK_DPAR_UPPER_OBJ_CUT_FINITE_TRH

Groups
Termination criteria

`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]

Example
`param = [ "MSK_DPAR_UPPER_OBJ_CUT_FINITE_TRH", "0.5e30" ]`

Groups
Termination criteria

13.5.2 Integer parameters

`MSK_IPAR_ANA_SOL_BASIS`

Controls whether the basis matrix is analyzed in solution analyzer.

Default
"ON"

Accepted
"ON", *"OFF"*

Example
`param = [ "MSK_IPAR_ANA_SOL_BASIS", "MSK_ON" ]`

Groups
Analysis

`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"*

Example
`param = [ "MSK_IPAR_ANA_SOL_PRINT_VIOLATED", "MSK_OFF" ]`

Groups
Analysis

`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"*

Example
`param = [ "MSK_IPAR_AUTO_SORT_A_BEFORE_OPT", "MSK_OFF" ]`

Groups

Debugging

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"*

Example

```
param = [ "MSK_IPAR_AUTO_UPDATE_SOL_INFO", "MSK_OFF" ]
```

Groups

Overall system

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"*

Example

```
param = [ "MSK_IPAR_BASIS_SOLVE_USE_PLUS_ONE", "MSK_OFF" ]
```

Groups

Simplex optimizer

MSK_IPAR_BI_CLEAN_OPTIMIZER

Controls which simplex optimizer is used in the clean-up phase. Anything else than *"MSK_OPTIMIZER_PRIMAL_SIMPLEX"* or *"MSK_OPTIMIZER_DUAL_SIMPLEX"* is equivalent to *"MSK_OPTIMIZER_FREE_SIMPLEX"*.

Default

"FREE"

Accepted

"FREE", *"INTPNT"*, *"CONIC"*, *"PRIMAL_SIMPLEX"*, *"DUAL_SIMPLEX"*, *"NEW_PRIMAL_SIMPLEX"*, *"NEW_DUAL_SIMPLEX"*, *"FREE_SIMPLEX"*, *"MIXED_INT"*

Example

```
param = [ "MSK_IPAR_BI_CLEAN_OPTIMIZER", "MSK_OPTIMIZER_FREE" ]
```

Groups

Basis identification, *Overall solver*

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"*

Example

```
param = [ "MSK_IPAR_BI_IGNORE_MAX_ITER", "MSK_OFF" ]
```

Groups

Interior-point method, *Basis identification*

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"*

Example
`param = [ "MSK_IPAR_BI_IGNORE_NUM_ERROR", "MSK_OFF" ]`

Groups
Interior-point method, *Basis identification*

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]

Example
`param = [ "MSK_IPAR_BI_MAX_ITERATIONS", "1000000" ]`

Groups
Basis identification, *Termination criteria*

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"*

Example
`param = [ "MSK_IPAR_CACHE_LICENSE", "MSK_ON" ]`

Groups
License manager

MSK_IPAR_COMPRESS_STATFILE

Control compression of stat files.

Default
"ON"

Accepted
"ON", *"OFF"*

Example
`param = [ "MSK_IPAR_COMPRESS_STATFILE", "MSK_ON" ]`

MSK_IPAR_FOLDING_USE

Controls whether and how to use problem folding (symmetry detection for continuous problems). Note that for symmetry detection for mixed-integer problems one should instead use the parameter *MSK_IPAR_MIO_SYMMETRY_LEVEL*.

Default

"FREE_UNLESS_BASIC"

Accepted

"OFF", *"FREE"*, *"FREE_UNLESS_BASIC"*, *"FORCE"*

Example

```
param = [ "MSK_IPAR_FOLDING_USE", "MSK_FOLDING_MODE_FREE_UNLESS_BASIC
" ]
```

Groups

Presolve

`MSK_IPAR_GETDUAL_CONVERT_LMIS`

Whether to perform LMI detection and optimization in the user-level dualizer.

Default

"ON"

Accepted

"ON", *"OFF"*

Example

```
param = [ "MSK_IPAR_GETDUAL_CONVERT_LMIS", "MSK_ON" ]
```

`MSK_IPAR_HEARTBEAT_SIM_FREQ_TICKS`

Controls how frequent the new simplex optimizer calls the user-defined callback function is called.

- -1. Logging is disabled.
- 0. Logging at highest frequency (every iteration).
- ≥ 1. Logging at given frequency measured in ticks.

Default

1000000

Accepted

[-1; +inf]

Example

```
param = [ "MSK_IPAR_HEARTBEAT_SIM_FREQ_TICKS", "1000000" ]
```

Groups

Simplex optimizer, *Output information*, *Logging*

`MSK_IPAR_INFEAS_GENERIC_NAMES`

Controls whether generic names are used when an infeasible subproblem is created.

Default

"OFF"

Accepted

"ON", *"OFF"*

Example

```
param = [ "MSK_IPAR_INFEAS_GENERIC_NAMES", "MSK_OFF" ]
```

Groups

Infeasibility report

`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"*

Example

```
param = [ "MSK_IPAR_INFEAS_REPORT_AUTO", "MSK_OFF" ]
```

Groups
Data input/output, *Solution input/output*

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]

Example
```
param = [ "MSK_IPAR_INFEAS_REPORT_LEVEL", "1" ]
```

Groups
Infeasibility report, *Output information*

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"*

Example
```
param = [ "MSK_IPAR_INTPNT_BASIS", "MSK_BI_ALWAYS" ]
```

See also
MSK_IPAR_BI_IGNORE_MAX_ITER, *MSK_IPAR_BI_IGNORE_NUM_ERROR*, *MSK_IPAR_BI_MAX_ITERATIONS*, *MSK_IPAR_BI_CLEAN_OPTIMIZER*

Groups
Interior-point method, *Basis identification*

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.

Example
```
param = [ "MSK_IPAR_INTPNT_DIFF_STEP", "MSK_ON" ]
```

Groups
Interior-point method

MSK_IPAR_INTPNT_HOTSTART

Currently not in use.

Default
"NONE"

Accepted
"NONE", *"PRIMAL"*, *"DUAL"*, *"PRIMAL_DUAL"*

Example
```
param = [ "MSK_IPAR_INTPNT_HOTSTART", "MSK_INTPNT_HOTSTART_NONE" ]
```

Groups
Interior-point method

MSK_IPAR_INTPNT_MAX_ITERATIONS

Controls the maximum number of iterations allowed in the interior-point optimizer.

Default
400

Accepted
[0; +inf]

Example
`param = [ "MSK_IPAR_INTPNT_MAX_ITERATIONS", "400" ]`

Groups
Interior-point method, *Termination criteria*

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]

Example
`param = [ "MSK_IPAR_INTPNT_MAX_NUM_COR", "-1" ]`

Groups
Interior-point method

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]

Example
`param = [ "MSK_IPAR_INTPNT_OFF_COL_TRH", "40" ]`

Groups
Interior-point method

MSK_IPAR_INTPNT_ORDER_GP_NUM_SEEDS

The GP ordering is dependent on a random seed. Therefore, trying several random seeds may lead to a better ordering. This parameter controls the number of random seeds tried.

A value of 0 means that MOSEK makes the choice.

Default
0

Accepted
[0; +inf]

Example
`param = [ "MSK_IPAR_INTPNT_ORDER_GP_NUM_SEEDS", "0" ]`

Groups
Interior-point method

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"*

Example
`param = [ "MSK_IPAR_INTPNT_ORDER_METHOD", "MSK_ORDER_METHOD_FREE" ]`

Groups
Interior-point method

MSK_IPAR_INTPNT_REGULARIZATION_USE

Controls whether regularization is allowed.

Default
"ON"

Accepted
"ON", *"OFF"*

Example
`param = [ "MSK_IPAR_INTPNT_REGULARIZATION_USE", "MSK_ON" ]`

Groups
Interior-point method

MSK_IPAR_INTPNT_SCALING

Controls how the problem is scaled before the interior-point optimizer is used.

Default
"FREE"

Accepted
"FREE", *"NONE"*

Example
`param = [ "MSK_IPAR_INTPNT_SCALING", "MSK_SCALING_FREE" ]`

Groups
Interior-point method

MSK_IPAR_INTPNT_SOLVE_FORM

Controls whether the primal or the dual problem is solved.

Default
"FREE"

Accepted
"FREE", *"PRIMAL"*, *"DUAL"*

Example
`param = [ "MSK_IPAR_INTPNT_SOLVE_FORM", "MSK_SOLVE_FREE" ]`

Groups
Interior-point method

MSK_IPAR_INTPNT_STARTING_POINT

Starting point used by the interior-point optimizer.

Default
"FREE"

Accepted
"FREE", *"GUESS"*, *"CONSTANT"*

Example

`param = [ "MSK_IPAR_INTPNT_STARTING_POINT", "MSK_STARTING_POINT_FREE" ]`

Groups

Interior-point method

`MSK_IPAR_LICENSE_DEBUG`

This option is used to turn on debugging of the license manager.

Default

"OFF"

Accepted

"ON", *"OFF"*

Example

`param = [ "MSK_IPAR_LICENSE_DEBUG", "MSK_OFF" ]`

Groups

License manager

`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]

Example

`param = [ "MSK_IPAR_LICENSE_PAUSE_TIME", "100" ]`

Groups

License manager

`MSK_IPAR_LICENSE_SUPPRESS_EXPIRE_WRNS`

Controls whether license features expire warnings are suppressed.

Default

"OFF"

Accepted

"ON", *"OFF"*

Example

`param = [ "MSK_IPAR_LICENSE_SUPPRESS_EXPIRE_WRNS", "MSK_OFF" ]`

Groups

License manager, *Output information*

`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]

Example

`param = [ "MSK_IPAR_LICENSE_TRH_EXPIRY_WRN", "7" ]`

Groups

License manager, *Output information*

`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"*

Example
`param = [ "MSK_IPAR_LICENSE_WAIT", "MSK_OFF" ]`

Groups
Overall solver, *Overall system*, *License manager*

`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]

Example
`param = [ "MSK_IPAR_LOG", "10" ]`

See also
MSK_IPAR_LOG_CUT_SECOND_OPT

Groups
Output information, *Logging*

`MSK_IPAR_LOG_ANA_PRO`

Controls amount of output from the problem analyzer.

Default
1

Accepted
[0; +inf]

Example
`param = [ "MSK_IPAR_LOG_ANA_PRO", "1" ]`

Groups
Analysis, *Logging*

`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]

Example
`param = [ "MSK_IPAR_LOG_BI", "1" ]`

Groups
Basis identification, *Output information*, *Logging*

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]

Example
`param = [ "MSK_IPAR_LOG_BI_FREQ", "2500" ]`

Groups
Basis identification, *Output information*, *Logging*

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]

Example
`param = [ "MSK_IPAR_LOG_CUT_SECOND_OPT", "1" ]`

See also
MSK_IPAR_LOG, *MSK_IPAR_LOG_INTPNT*, *MSK_IPAR_LOG_MIO*, *MSK_IPAR_LOG_SIM*

Groups
Output information, *Logging*

MSK_IPAR_LOG_EXPAND

Controls the amount of logging when a data item such as the maximum number constrains is expanded.

Default
1

Accepted
[0; +inf]

Example
`param = [ "MSK_IPAR_LOG_EXPAND", "1" ]`

Groups
Output information, *Logging*

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]

Example
`param = [ "MSK_IPAR_LOG_FEAS_REPAIR", "1" ]`

Groups
Output information, *Logging*

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]

Example
`param = [ "MSK_IPAR_LOG_FILE", "1" ]`

Groups
Data input/output, *Output information*, *Logging*

MSK_IPAR_LOG_INCLUDE_SUMMARY

Not relevant for this API.

Default
"OFF"

Accepted
"ON", *"OFF"*

Example
`param = [ "MSK_IPAR_LOG_INCLUDE_SUMMARY", "MSK_OFF" ]`

Groups
Output information, *Logging*

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]

Example
`param = [ "MSK_IPAR_LOG_INFEAS_ANA", "1" ]`

Groups
Infeasibility report, *Output information*, *Logging*

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]

Example
`param = [ "MSK_IPAR_LOG_INTPNT", "1" ]`

Groups
Interior-point method, *Output information*, *Logging*

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"*

Example
`param = [ "MSK_IPAR_LOG_LOCAL_INFO", "MSK_ON" ]`

Groups
Output information, *Logging*

`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]

Example
`param = [ "MSK_IPAR_LOG_MIO", "4" ]`

Groups
Mixed-integer optimization, *Output information*, *Logging*

`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]

Example
`param = [ "MSK_IPAR_LOG_MIO_FREQ", "10" ]`

Groups
Mixed-integer optimization, *Output information*, *Logging*

`MSK_IPAR_LOG_ORDER`

If turned on, then factor lines are added to the log.

Default
1

Accepted
[0; +inf]

Example
`param = [ "MSK_IPAR_LOG_ORDER", "1" ]`

Groups
Output information, *Logging*

`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]

Example
`param = [ "MSK_IPAR_LOG_PRESOLVE", "1" ]`

Groups
Logging

`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]

Example
`param = [ "MSK_IPAR_LOG_SENSITIVITY", "1" ]`

Groups
Output information, *Logging*

`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]

Example
`param = [ "MSK_IPAR_LOG_SENSITIVITY_OPT", "0" ]`

Groups
Output information, *Logging*

`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]

Example
`param = [ "MSK_IPAR_LOG_SIM", "4" ]`

Groups
Simplex optimizer, *Output information*, *Logging*

`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]

Example
`param = [ "MSK_IPAR_LOG_SIM_FREQ", "1000" ]`

Groups
Simplex optimizer, *Output information*, *Logging*

MSK_IPAR_LOG_SIM_FREQ_GIGA_TICKS

Controls how frequent the new simplex optimizer outputs information about the optimization and how frequent the user-defined callback function is called.

- -1. Logging is disabled.
- 0. Logging at highest frequency (every iteration).
- ≥ 1. Logging at given frequency measured in giga ticks.

Default
100

Accepted
[-1; +inf]

Example
`param = [ "MSK_IPAR_LOG_SIM_FREQ_GIGA_TICKS", "100" ]`

Groups
Simplex optimizer, *Output information*, *Logging*

MSK_IPAR_LOG_STORAGE

When turned on, **MOSEK** prints messages regarding the storage usage and allocation.

Default
0

Accepted
[0; +inf]

Example
`param = [ "MSK_IPAR_LOG_STORAGE", "0" ]`

Groups
Output information, *Overall system*, *Logging*

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]

Example
`param = [ "MSK_IPAR_MAX_NUM_WARNINGS", "10" ]`

Groups
Output information

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"*

Example
`param = [ "MSK_IPAR_MIO_BRANCH_DIR", "MSK_BRANCH_DIR_FREE" ]`

Groups
Mixed-integer optimization

MSK_IPAR_MIO_CONFLICT_ANALYSIS_LEVEL

Controls the amount of conflict analysis employed by the mixed-integer optimizer.

- −1. The optimizer chooses the level of conflict analysis employed
- 0. conflict analysis is disabled
- 1. A lower amount of conflict analysis is employed
- 2. A higher amount of conflict analysis is employed

Default
-1

Accepted
[-1; 2]

Example
`param = [ "MSK_IPAR_MIO_CONFLICT_ANALYSIS_LEVEL", "-1" ]`

Groups
Mixed-integer optimization

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"*

Example
`param = [ "MSK_IPAR_MIO_CONIC_OUTER_APPROXIMATION", "MSK_OFF" ]`

Groups
Mixed-integer optimization

MSK_IPAR_MIO_CONSTRUCT_SOL

If set to *"MSK_ON"* and all integer variables have been given a value for which a feasible mixed integer solution exists, then **MOSEK** generates an initial solution to the mixed integer problem by fixing all integer values and solving the remaining problem.

Default
"OFF"

Accepted
"ON", *"OFF"*

Example
`param = [ "MSK_IPAR_MIO_CONSTRUCT_SOL", "MSK_OFF" ]`

Groups
Mixed-integer optimization

MSK_IPAR_MIO_CROSSOVER_MAX_NODES

Controls the maximum number of nodes allowed in each call to the Crossover heuristic. The default value of -1 means that the value is determined automatically. A value of zero turns off the heuristic.

Default
-1

Accepted
[-1; +inf]

Example
`param = [ "MSK_IPAR_MIO_CROSSOVER_MAX_NODES", "-1" ]`

Groups
Mixed-integer optimization

`MSK_IPAR_MIO_CUT_CLIQUE`

Controls whether clique cuts should be generated.

Default
"ON"

Accepted
"ON", *"OFF"*

Example
`param = [ "MSK_IPAR_MIO_CUT_CLIQUE", "MSK_ON" ]`

Groups
Mixed-integer optimization

`MSK_IPAR_MIO_CUT_CMIR`

Controls whether mixed integer rounding cuts should be generated.

Default
"ON"

Accepted
"ON", *"OFF"*

Example
`param = [ "MSK_IPAR_MIO_CUT_CMIR", "MSK_ON" ]`

Groups
Mixed-integer optimization

`MSK_IPAR_MIO_CUT_GMI`

Controls whether GMI cuts should be generated.

Default
"ON"

Accepted
"ON", *"OFF"*

Example
`param = [ "MSK_IPAR_MIO_CUT_GMI", "MSK_ON" ]`

Groups
Mixed-integer optimization

`MSK_IPAR_MIO_CUT_IMPLIED_BOUND`

Controls whether implied bound cuts should be generated.

Default
"ON"

Accepted
"ON", *"OFF"*

Example
`param = [ "MSK_IPAR_MIO_CUT_IMPLIED_BOUND", "MSK_ON" ]`

Groups
Mixed-integer optimization

`MSK_IPAR_MIO_CUT_KNAPSACK_COVER`

Controls whether knapsack cover cuts should be generated.

Default
"ON"

Accepted
"ON", *"OFF"*

Example
`param = [ "MSK_IPAR_MIO_CUT_KNAPSACK_COVER", "MSK_ON" ]`

Groups
Mixed-integer optimization

MSK_IPAR_MIO_CUT_LIPRO

Controls whether lift-and-project cuts should be generated.

Default
"OFF"

Accepted
"ON", *"OFF"*

Example
`param = [ "MSK_IPAR_MIO_CUT_LIPRO", "MSK_OFF" ]`

Groups
Mixed-integer optimization

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]

Example
`param = [ "MSK_IPAR_MIO_CUT_SELECTION_LEVEL", "-1" ]`

Groups
Mixed-integer optimization

MSK_IPAR_MIO_DATA_PERMUTATION_METHOD

Controls what problem data permutation method is appplied to mixed-integer problems.

Default
"NONE"

Accepted
"NONE", *"CYCLIC_SHIFT"*, *"RANDOM"*

Example
```
param = [ "MSK_IPAR_MIO_DATA_PERMUTATION_METHOD",
"MSK_MIO_DATA_PERMUTATION_METHOD_NONE" ]
```

Groups
Mixed-integer optimization

MSK_IPAR_MIO_DUAL_RAY_ANALYSIS_LEVEL

Controls the amount of dual ray analysis employed by the mixed-integer optimizer.

- -1. The optimizer chooses the level of dual ray analysis employed
- 0. Dual ray analysis is disabled
- 1. A lower amount of dual ray analysis is employed
- 2. A higher amount of dual ray analysis is employed

Default
-1

Accepted
[-1; 2]

Example
`param = [ "MSK_IPAR_MIO_DUAL_RAY_ANALYSIS_LEVEL", "-1" ]`

Groups
Mixed-integer optimization

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]

Example
```
param = [ "MSK_IPAR_MIO_FEASPUMP_LEVEL", "-1" ]
```

Groups
Mixed-integer optimization

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]

Example
```
param = [ "MSK_IPAR_MIO_HEURISTIC_LEVEL", "-1" ]
```

Groups
Mixed-integer optimization

MSK_IPAR_MIO_INDEPENDENT_BLOCK_LEVEL

Controls the way the mixed-integer optimizer tries to find and exploit a decomposition of the problem into independent blocks.

- -1. The optimizer chooses how independent-block structure is handled
- 0. No independent-block structure is detected
- 1. Independent-block structure may be exploited only in presolve
- 2. Independent-block structure may be exploited through a dedicated algorithm after the root node
- 3. Independent-block structure may be exploited through a dedicated algorithm before the root node

Default
-1

Accepted
[-1; 3]

Example
```
param = [ "MSK_IPAR_MIO_INDEPENDENT_BLOCK_LEVEL", "-1" ]
```

Groups
Mixed-integer optimization

`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]

Example
`param = [ "MSK_IPAR_MIO_MAX_NUM_BRANCHES", "-1" ]`

Groups
Mixed-integer optimization, *Termination criteria*

`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]

Example
`param = [ "MSK_IPAR_MIO_MAX_NUM_RELAXS", "-1" ]`

Groups
Mixed-integer optimization

`MSK_IPAR_MIO_MAX_NUM_RESTARTS`

Maximum number of restarts allowed during the branch and bound search.

Default
10

Accepted
[0; +inf]

Example
`param = [ "MSK_IPAR_MIO_MAX_NUM_RESTARTS", "10" ]`

Groups
Mixed-integer optimization

`MSK_IPAR_MIO_MAX_NUM_ROOT_CUT_ROUNDS`

Maximum number of cut separation rounds at the root node.

Default
100

Accepted
[0; +inf]

Example
`param = [ "MSK_IPAR_MIO_MAX_NUM_ROOT_CUT_ROUNDS", "100" ]`

Groups
Mixed-integer optimization, *Termination criteria*

`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]

Example

`param = [ "MSK_IPAR_MIO_MAX_NUM_SOLUTIONS", "-1" ]`

Groups

Mixed-integer optimization, *Termination criteria*

`MSK_IPAR_MIO_MEMORY_EMPHASIS_LEVEL`

Controls how much emphasis is put on reducing memory usage. Being more conservative about memory usage may come at the cost of decreased solution speed.

- 0. The optimizer chooses
- 1. More emphasis is put on reducing memory usage and less on speed

Default

0

Accepted

[0; +1]

Example

`param = [ "MSK_IPAR_MIO_MEMORY_EMPHASIS_LEVEL", "0" ]`

Groups

Mixed-integer optimization

`MSK_IPAR_MIO_MIN_REL`

Number of times a variable must have been branched on for its pseudocost to be considered reliable.

Default

5

Accepted

[0; +inf]

Example

`param = [ "MSK_IPAR_MIO_MIN_REL", "5" ]`

Groups

Mixed-integer optimization

`MSK_IPAR_MIO_MODE`

Controls whether the optimizer includes the integer restrictions and disjunctive constraints when solving a (mixed) integer optimization problem.

Default

"SATISFIED"

Accepted

"IGNORED", *"SATISFIED"*

Example

`param = [ "MSK_IPAR_MIO_MODE", "MSK_MIO_MODE_SATISFIED" ]`

Groups

Overall solver

`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"*, *"NEW_PRIMAL_SIMPLEX"*, *"NEW_DUAL_SIMPLEX"*, *"FREE_SIMPLEX"*, *"MIXED_INT"*

Example

`param = [ "MSK_IPAR_MIO_NODE_OPTIMIZER", "MSK_OPTIMIZER_FREE" ]`

Groups

Mixed-integer optimization

MSK_IPAR_MIO_NODE_SELECTION

Controls the node selection strategy employed by the mixed-integer optimizer.

Default
"FREE"
Accepted
"FREE", *"FIRST"*, *"BEST"*, *"PSEUDO"*
Example
`param = [ "MSK_IPAR_MIO_NODE_SELECTION", "MSK_MIO_NODE_SELECTION_FREE" ]`
Groups
Mixed-integer optimization

MSK_IPAR_MIO_NUMERICAL_EMPHASIS_LEVEL

Controls how much emphasis is put on reducing numerical problems possibly at the expense of solution speed.

- 0. The optimizer chooses
- 1. More emphasis is put on reducing numerical problems
- 2. Even more emphasis

Default
0
Accepted
[0; +2]
Example
`param = [ "MSK_IPAR_MIO_NUMERICAL_EMPHASIS_LEVEL", "0" ]`
Groups
Mixed-integer optimization

MSK_IPAR_MIO_OPT_FACE_MAX_NODES

Controls the maximum number of nodes allowed in each call to the optimal face heuristic. The default value of -1 means that the value is determined automatically. A value of zero turns off the heuristic.

Default
-1
Accepted
[-1; +inf]
Example
`param = [ "MSK_IPAR_MIO_OPT_FACE_MAX_NODES", "-1" ]`
Groups
Mixed-integer optimization

MSK_IPAR_MIO_PERSPECTIVE_REFORMULATE

Enables or disables perspective reformulation in presolve.

Default
"ON"
Accepted
"ON", *"OFF"*
Example
`param = [ "MSK_IPAR_MIO_PERSPECTIVE_REFORMULATE", "MSK_ON" ]`
Groups
Mixed-integer optimization

MSK_IPAR_MIO_PRESOLVE_AGGREGATOR_USE

Controls if the aggregator should be used.

Default
"ON"

Accepted
"ON", *"OFF"*

Example
`param = [ "MSK_IPAR_MIO_PRESOLVE_AGGREGATOR_USE", "MSK_ON" ]`

Groups
Presolve

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]

Example
`param = [ "MSK_IPAR_MIO_PROBING_LEVEL", "-1" ]`

Groups
Mixed-integer optimization

MSK_IPAR_MIO_PROPAGATE_OBJECTIVE_CONSTRAINT

Use objective domain propagation.

Default
"OFF"

Accepted
"ON", *"OFF"*

Example
`param = [ "MSK_IPAR_MIO_PROPAGATE_OBJECTIVE_CONSTRAINT", "MSK_OFF" ]`

Groups
Mixed-integer optimization

MSK_IPAR_MIO_QCQO_REFORMULATION_METHOD

Controls what reformulation method is applied to mixed-integer quadratic problems.

Default
"FREE"

Accepted
"FREE", *"NONE"*, *"LINEARIZATION"*, *"EIGEN_VAL_METHOD"*, *"DIAG_SDP"*, *"RELAX_SDP"*

Example
```
param = [ "MSK_IPAR_MIO_QCQO_REFORMULATION_METHOD",
"MSK_MIO_QCQO_REFORMULATION_METHOD_FREE" ]
```

Groups
Mixed-integer optimization

MSK_IPAR_MIO_RENS_MAX_NODES

Controls the maximum number of nodes allowed in each call to the RENS heuristic. The default value of -1 means that the value is determined automatically. A value of zero turns off the heuristic.

Default
-1

Accepted
[-1; +inf]

Example
`param = [ "MSK_IPAR_MIO_RENS_MAX_NODES", "-1" ]`

Groups
Mixed-integer optimization

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]

Example
`param = [ "MSK_IPAR_MIO_RINS_MAX_NODES", "-1" ]`

Groups
Mixed-integer optimization

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"*, *"NEW_PRIMAL_SIMPLEX"*, *"NEW_DUAL_SIMPLEX"*, *"FREE_SIMPLEX"*, *"MIXED_INT"*

Example
`param = [ "MSK_IPAR_MIO_ROOT_OPTIMIZER", "MSK_OPTIMIZER_FREE" ]`

Groups
Mixed-integer optimization

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]

Example
`param = [ "MSK_IPAR_MIO_SEED", "42" ]`

Groups
Mixed-integer optimization

MSK_IPAR_MIO_SYMMETRY_LEVEL

Controls the amount of symmetry detection and handling employed by the mixed-integer optimizer in presolve.

- -1. The optimizer chooses the level of symmetry detection and handling employed
- 0. Symmetry detection and handling is disabled

- 1. A low amount of symmetry detection and handling is employed
- 2. A medium amount of symmetry detection and handling is employed
- 3. A high amount of symmetry detection and handling is employed
- 4. An extremely high amount of symmetry detection and handling is employed

Default
-1
Accepted
[-1; 4]
Example
`param = [ "MSK_IPAR_MIO_SYMMETRY_LEVEL", "-1" ]`
Groups
Mixed-integer optimization

`MSK_IPAR_MIO_VAR_SELECTION`

Controls the variable selection strategy employed by the mixed-integer optimizer.

Default
"FREE"
Accepted
"FREE", *"PSEUDOCOST"*, *"STRONG"*
Example
`param = [ "MSK_IPAR_MIO_VAR_SELECTION", "MSK_MIO_VAR_SELECTION_FREE" ]`
Groups
Mixed-integer optimization

`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]
Example
`param = [ "MSK_IPAR_MIO_VB_DETECTION_LEVEL", "-1" ]`
Groups
Mixed-integer optimization

`MSK_IPAR_MT_SPINCOUNT`

Set the number of iterations to spin before sleeping.

Default
0
Accepted
[0; 1000000000]
Example
`param = [ "MSK_IPAR_MT_SPINCOUNT", "0" ]`
Groups
Overall system

MSK_IPAR_NG

Not in use.

Default

"OFF"

Accepted

"ON", *"OFF"*

Example

`param = [ "MSK_IPAR_NG", "MSK_OFF" ]`

MSK_IPAR_NUM_THREADS

Controls the number of threads employed by the optimizer. If set to 0 then the number of threads is chosen by the optimizer, typically as minimum(number of cores, 32). If set to a positive value then exactly the selected number of threads will be used.

Default

0

Accepted

[0; +inf]

Example

`param = [ "MSK_IPAR_NUM_THREADS", "0" ]`

Groups

Overall system

MSK_IPAR_OPF_WRITE_HEADER

Write a text header with date and **MOSEK** version in an OPF file.

Default

"ON"

Accepted

"ON", *"OFF"*

Example

`param = [ "MSK_IPAR_OPF_WRITE_HEADER", "MSK_ON" ]`

Groups

Data input/output

MSK_IPAR_OPF_WRITE_HINTS

Write a hint section with problem dimensions in the beginning of an OPF file.

Default

"ON"

Accepted

"ON", *"OFF"*

Example

`param = [ "MSK_IPAR_OPF_WRITE_HINTS", "MSK_ON" ]`

Groups

Data input/output

MSK_IPAR_OPF_WRITE_LINE_LENGTH

Aim to keep lines in OPF files not much longer than this.

Default

80

Accepted

[0; +inf]

Example

`param = [ "MSK_IPAR_OPF_WRITE_LINE_LENGTH", "80" ]`

Groups

Data input/output

MSK_IPAR_OPF_WRITE_PARAMETERS

Write a parameter section in an OPF file.

Default

"OFF"

Accepted

"ON", *"OFF"*

Example

```
param = [ "MSK_IPAR_OPF_WRITE_PARAMETERS", "MSK_OFF" ]
```

Groups

Data input/output

MSK_IPAR_OPF_WRITE_PROBLEM

Write objective, constraints, bounds etc. to an OPF file.

Default

"ON"

Accepted

"ON", *"OFF"*

Example

```
param = [ "MSK_IPAR_OPF_WRITE_PROBLEM", "MSK_ON" ]
```

Groups

Data input/output

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"*

Example

```
param = [ "MSK_IPAR_OPF_WRITE_SOL_BAS", "MSK_ON" ]
```

Groups

Data input/output

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"*

Example

```
param = [ "MSK_IPAR_OPF_WRITE_SOL_ITG", "MSK_ON" ]
```

Groups

Data input/output

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"*

Example

```
param = [ "MSK_IPAR_OPF_WRITE_SOL_ITR", "MSK_ON" ]
```

Groups

Data input/output

`MSK_IPAR_OPF_WRITE_SOLUTIONS`

Enable inclusion of solutions in the OPF files.

Default

"OFF"

Accepted

"ON", *"OFF"*

Example

```
param = [ "MSK_IPAR_OPF_WRITE_SOLUTIONS", "MSK_OFF" ]
```

Groups

Data input/output

`MSK_IPAR_OPTIMIZER`

The parameter controls which optimizer is used to optimize the task.

Default

"FREE"

Accepted

"FREE", *"INTPNT"*, *"CONIC"*, *"PRIMAL_SIMPLEX"*, *"DUAL_SIMPLEX"*, *"NEW_PRIMAL_SIMPLEX"*, *"NEW_DUAL_SIMPLEX"*, *"FREE_SIMPLEX"*, *"MIXED_INT"*

Example

```
param = [ "MSK_IPAR_OPTIMIZER", "MSK_OPTIMIZER_FREE" ]
```

Groups

Overall solver

`MSK_IPAR_PARAM_READ_CASE_NAME`

If turned on, then names in the parameter file are case sensitive.

Default

"ON"

Accepted

"ON", *"OFF"*

Example

```
param = [ "MSK_IPAR_PARAM_READ_CASE_NAME", "MSK_ON" ]
```

Groups

Data input/output

`MSK_IPAR_PARAM_READ_IGN_ERROR`

If turned on, then errors in parameter settings is ignored.

Default

"OFF"

Accepted

"ON", *"OFF"*

Example

```
param = [ "MSK_IPAR_PARAM_READ_IGN_ERROR", "MSK_OFF" ]
```

Groups

Data input/output

`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]

Example

`param = [ "MSK_IPAR_PRESOLVE_ELIMINATOR_MAX_FILL", "-1" ]`

Groups

Presolve

`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]

Example

`param = [ "MSK_IPAR_PRESOLVE_ELIMINATOR_MAX_NUM_TRIES", "-1" ]`

Groups

Presolve

`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]

Example

`param = [ "MSK_IPAR_PRESOLVE_LINDEP_ABS_WORK_TRH", "100" ]`

Groups

Presolve

`MSK_IPAR_PRESOLVE_LINDEP_NEW`

Controls whether a new experimental linear dependency checker is employed.

Default

"OFF"

Accepted

"ON", *"OFF"*

Example

`param = [ "MSK_IPAR_PRESOLVE_LINDEP_NEW", "MSK_OFF" ]`

Groups

Presolve

`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]

Example

`param = [ "MSK_IPAR_PRESOLVE_LINDEP_REL_WORK_TRH", "100" ]`

Groups

Presolve

`MSK_IPAR_PRESOLVE_LINDEP_USE`

Controls whether the linear constraints are checked for linear dependencies.

Default
"ON"

Accepted
"ON", *"OFF"*

Example
```
param = [ "MSK_IPAR_PRESOLVE_LINDEP_USE", "MSK_ON" ]
```

Groups
Presolve

`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]

Example
```
param = [ "MSK_IPAR_PRESOLVE_MAX_NUM_PASS", "-1" ]
```

Groups
Presolve

`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]

Example
```
param = [ "MSK_IPAR_PRESOLVE_MAX_NUM_REDUCTIONS", "-1" ]
```

Groups
Overall solver, *Presolve*

`MSK_IPAR_PRESOLVE_USE`

Controls whether the presolve is applied to a problem before it is optimized.

Default
"FREE"

Accepted
"OFF", *"ON"*, *"FREE"*

Example
```
param = [ "MSK_IPAR_PRESOLVE_USE", "MSK_PRESOLVE_MODE_FREE" ]
```

Groups
Overall solver, *Presolve*

`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"*, *"NEW_PRIMAL_SIMPLEX"*, *"NEW_DUAL_SIMPLEX"*, *"FREE_SIMPLEX"*, *"MIXED_INT"*

Example

```
param = [ "MSK_IPAR_PRIMAL_REPAIR_OPTIMIZER", "MSK_OPTIMIZER_FREE" ]
```

Groups

Overall solver

`MSK_IPAR_PTF_WRITE_PARAMETERS`

If *MSK_IPAR_PTF_WRITE_PARAMETERS* is *"MSK_ON"*, the parameters section is written.

Default

"OFF"

Accepted

"ON", *"OFF"*

Example

```
param = [ "MSK_IPAR_PTF_WRITE_PARAMETERS", "MSK_OFF" ]
```

Groups

Data input/output

`MSK_IPAR_PTF_WRITE_SINGLE_PSD_TERMS`

Controls whether PSD terms with a coefficient matrix of just one non-zero are written as a single term instead of as a matrix term.

Default

"OFF"

Accepted

"ON", *"OFF"*

Example

```
param = [ "MSK_IPAR_PTF_WRITE_SINGLE_PSD_TERMS", "MSK_OFF" ]
```

Groups

Data input/output

`MSK_IPAR_PTF_WRITE_SOLUTIONS`

If *MSK_IPAR_PTF_WRITE_SOLUTIONS* is *"MSK_ON"*, the solution section is written if any solutions are available, otherwise solution section is not written even if solutions are available.

Default

"OFF"

Accepted

"ON", *"OFF"*

Example

```
param = [ "MSK_IPAR_PTF_WRITE_SOLUTIONS", "MSK_OFF" ]
```

Groups

Data input/output

`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"*

Example

```
param = [ "MSK_IPAR_PTF_WRITE_TRANSFORM", "MSK_ON" ]
```

Groups

Data input/output

MSK_IPAR_READ_ASYNC

Controls whether files are read using synchronous or asynchronous reader.

Default
"OFF"

Accepted
- *"ON"*: Use asynchronous reader
- *"OFF"*: Use synchronous reader

Example
```
param = [ "MSK_IPAR_READ_ASYNC", "MSK_OFF" ]
```

Groups
Data input/output

MSK_IPAR_READ_DEBUG

Turns on additional debugging information when reading files.

Default
"OFF"

Accepted
"ON", *"OFF"*

Example
```
param = [ "MSK_IPAR_READ_DEBUG", "MSK_OFF" ]
```

Groups
Data input/output

MSK_IPAR_READ_KEEP_FREE_CON

Controls whether the free constraints are included in the problem. Applies to MPS files.

Default
"OFF"

Accepted
- *"ON"*: The free constraints are kept.
- *"OFF"*: The free constraints are discarded.

Example
```
param = [ "MSK_IPAR_READ_KEEP_FREE_CON", "MSK_OFF" ]
```

Groups
Data input/output

MSK_IPAR_READ_MPS_FORMAT

Controls how strictly the MPS file reader interprets the MPS format.

Default
"FREE"

Accepted
"STRICT", *"RELAXED"*, *"FREE"*, *"CPLEX"*

Example
```
param = [ "MSK_IPAR_READ_MPS_FORMAT", "MSK_MPS_FORMAT_FREE" ]
```

Groups
Data input/output

MSK_IPAR_READ_MPS_WIDTH

Controls the maximal number of characters allowed in one line of the MPS file.

Default
1024

Accepted
[80; +inf]

Example

param = ["MSK_IPAR_READ_MPS_WIDTH", "1024"]

Groups

Data input/output

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"*

Example

param = ["MSK_IPAR_READ_TASK_IGNORE_PARAM", "MSK_OFF"]

Groups

Data input/output

MSK_IPAR_REMOTE_USE_COMPRESSION

Use compression when sending data to an optimization server.

Default

"ZSTD"

Accepted

"NONE", *"FREE"*, *"GZIP"*, *"ZSTD"*

Example

param = ["MSK_IPAR_REMOTE_USE_COMPRESSION", "MSK_COMPRESS_ZSTD"]

MSK_IPAR_REMOVE_UNUSED_SOLUTIONS

Removes unused solutions before the optimization is performed.

Default

"OFF"

Accepted

"ON", *"OFF"*

Example

param = ["MSK_IPAR_REMOVE_UNUSED_SOLUTIONS", "MSK_OFF"]

Groups

Overall system

MSK_IPAR_SENSITIVITY_ALL

Not applicable.

Default

"OFF"

Accepted

"ON", *"OFF"*

Example

param = ["MSK_IPAR_SENSITIVITY_ALL", "MSK_OFF"]

Groups

Overall solver

MSK_IPAR_SENSITIVITY_TYPE

Controls which type of sensitivity analysis is to be performed.

Default

"BASIS"

Accepted

"BASIS"

Example

`param = [ "MSK_IPAR_SENSITIVITY_TYPE", "MSK_SENSITIVITY_TYPE_BASIS" ]`

Groups

Overall solver

`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"*

Example

`param = [ "MSK_IPAR_SIM_BASIS_FACTOR_USE", "MSK_ON" ]`

Groups

Simplex optimizer

`MSK_IPAR_SIM_DEGEN`

Controls how aggressively degeneration is handled.

Default

"FREE"

Accepted

"NONE", *"FREE"*, *"AGGRESSIVE"*, *"MODERATE"*, *"MINIMUM"*

Example

`param = [ "MSK_IPAR_SIM_DEGEN", "MSK_SIM_DEGEN_FREE" ]`

Groups

Simplex optimizer

`MSK_IPAR_SIM_DETECT_PWL`

Not in use.

Default

"ON"

Accepted

- *"ON"*: PWL are detected.
- *"OFF"*: PWL are not detected.

Example

`param = [ "MSK_IPAR_SIM_DETECT_PWL", "MSK_ON" ]`

Groups

Simplex optimizer

`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]

Example

`param = [ "MSK_IPAR_SIM_DUAL_CRASH", "90" ]`

Groups

Dual simplex

`MSK_IPAR_SIM_DUAL_PHASEONE_METHOD`

An experimental feature.

Default
0

Accepted
[0; 10]

Example
`param = [ "MSK_IPAR_SIM_DUAL_PHASEONE_METHOD", "0" ]`

Groups
Simplex optimizer

`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]

Example
`param = [ "MSK_IPAR_SIM_DUAL_RESTRICT_SELECTION", "50" ]`

Groups
Dual simplex

`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"*

Example
`param = [ "MSK_IPAR_SIM_DUAL_SELECTION", "MSK_SIM_SELECTION_FREE" ]`

Groups
Dual simplex

`MSK_IPAR_SIM_EXPLOIT_DUPVEC`

Controls if the simplex optimizers are allowed to exploit duplicated columns.

Default
"OFF"

Accepted
"ON", *"OFF"*, *"FREE"*

Example
`param = [ "MSK_IPAR_SIM_EXPLOIT_DUPVEC", "MSK_SIM_EXPLOIT_DUPVEC_OFF" ]`

Groups
Simplex optimizer

MSK_IPAR_SIM_HOTSTART

Controls the type of hot-start that the simplex optimizer perform.

Default
"FREE"

Accepted
"NONE", *"FREE"*, *"STATUS_KEYS"*

Example
`param = [ "MSK_IPAR_SIM_HOTSTART", "MSK_SIM_HOTSTART_FREE" ]`

Groups
Simplex optimizer

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.

Example
`param = [ "MSK_IPAR_SIM_HOTSTART_LU", "MSK_ON" ]`

Groups
Simplex optimizer

MSK_IPAR_SIM_MAX_ITERATIONS

Maximum number of iterations that can be used by a simplex optimizer.

Default
10000000

Accepted
[0; +inf]

Example
`param = [ "MSK_IPAR_SIM_MAX_ITERATIONS", "10000000" ]`

Groups
Simplex optimizer, *Termination criteria*

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]

Example
`param = [ "MSK_IPAR_SIM_MAX_NUM_SETBACKS", "250" ]`

Groups
Simplex optimizer

MSK_IPAR_SIM_NON_SINGULAR

Controls if the simplex optimizer ensures a non-singular basis, if possible.

Default
"ON"

Accepted
"ON", *"OFF"*

Example

`param = [ "MSK_IPAR_SIM_NON_SINGULAR", "MSK_ON" ]`

Groups

Simplex optimizer

MSK_IPAR_SIM_PRECISION

Experimental. Usage not recommended.

Default

"NORMAL"

Accepted

"NORMAL", *"EXTENDED"*

Example

`param = [ "MSK_IPAR_SIM_PRECISION", "MSK_SIM_PRECISION_NORMAL" ]`

Groups

Overall solver

MSK_IPAR_SIM_PRECISION_BOOST

Controls whether the simplex optimizer is allowed to boost the precision during the computations if possible.

Default

"OFF"

Accepted

"ON", *"OFF"*

Example

`param = [ "MSK_IPAR_SIM_PRECISION_BOOST", "MSK_OFF" ]`

Groups

Simplex optimizer

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]

Example

`param = [ "MSK_IPAR_SIM_PRIMAL_CRASH", "90" ]`

Groups

Primal simplex

MSK_IPAR_SIM_PRIMAL_PHASEONE_METHOD

An experimental feature.

Default

0

Accepted

[0; 10]

Example

`param = [ "MSK_IPAR_SIM_PRIMAL_PHASEONE_METHOD", "0" ]`

Groups

Simplex optimizer

`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]

Example
`param = [ "MSK_IPAR_SIM_PRIMAL_RESTRICT_SELECTION", "50" ]`

Groups
Primal simplex

`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"*

Example
`param = [ "MSK_IPAR_SIM_PRIMAL_SELECTION", "MSK_SIM_SELECTION_FREE" ]`

Groups
Primal simplex

`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]

Example
`param = [ "MSK_IPAR_SIM_REFACTOR_FREQ", "0" ]`

Groups
Simplex optimizer

`MSK_IPAR_SIM_REFORMULATION`

Controls if the simplex optimizers are allowed to reformulate the problem.

Default
"OFF"

Accepted
"ON", *"OFF"*, *"FREE"*, *"AGGRESSIVE"*

Example
`param = [ "MSK_IPAR_SIM_REFORMULATION", "MSK_SIM_REFORMULATION_OFF" ]`

Groups
Simplex optimizer

`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"*

Example
`param = [ "MSK_IPAR_SIM_SAVE_LU", "MSK_OFF" ]`

Groups
Simplex optimizer

`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"*

Example
`param = [ "MSK_IPAR_SIM_SCALING", "MSK_SCALING_FREE" ]`

Groups
Simplex optimizer

`MSK_IPAR_SIM_SCALING_METHOD`

Controls how the problem is scaled before a simplex optimizer is used.

Default
"POW2"

Accepted
"POW2", *"FREE"*

Example
`param = [ "MSK_IPAR_SIM_SCALING_METHOD", "MSK_SCALING_METHOD_POW2" ]`

Groups
Simplex optimizer

`MSK_IPAR_SIM_SEED`

Sets the random seed used for randomization in the simplex optimizers.

Default
23456

Accepted
[0; 32749]

Example
`param = [ "MSK_IPAR_SIM_SEED", "23456" ]`

Groups
Simplex optimizer

`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"*

Example
`param = [ "MSK_IPAR_SIM_SOLVE_FORM", "MSK_SOLVE_FREE" ]`

Groups
Simplex optimizer

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"*

Example
`param = [ "MSK_IPAR_SIM_SWITCH_OPTIMIZER", "MSK_OFF" ]`

Groups
Simplex optimizer

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"*

Example
`param = [ "MSK_IPAR_SOL_FILTER_KEEP_BASIC", "MSK_OFF" ]`

Groups
Solution input/output

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]

Example
`param = [ "MSK_IPAR_SOL_READ_NAME_WIDTH", "-1" ]`

Groups
Data input/output, *Solution input/output*

MSK_IPAR_SOL_READ_WIDTH

Controls the maximal acceptable width of line in the solutions when read by **MOSEK**.

Default
1024

Accepted
[80; +inf]

Example
`param = [ "MSK_IPAR_SOL_READ_WIDTH", "1024" ]`

Groups
Data input/output, *Solution input/output*

`MSK_IPAR_TIMING_LEVEL`

Controls the amount of timing performed inside **MOSEK**.

Default
1

Accepted
[0; +inf]

Example
`param = [ "MSK_IPAR_TIMING_LEVEL", "1" ]`

Groups
Overall system

`MSK_IPAR_WRITE_ASYNC`

Controls whether files are read using synchronous or asynchronous writer.

Default
"OFF"

Accepted

- *"ON"*: Use asynchronous writer
- *"OFF"*: Use synchronous writer

Example
`param = [ "MSK_IPAR_WRITE_ASYNC", "MSK_OFF" ]`

Groups
Data input/output

`MSK_IPAR_WRITE_BAS_CONSTRAINTS`

Controls whether the constraint section is written to the basic solution file.

Default
"ON"

Accepted
"ON", *"OFF"*

Example
`param = [ "MSK_IPAR_WRITE_BAS_CONSTRAINTS", "MSK_ON" ]`

Groups
Data input/output, *Solution input/output*

`MSK_IPAR_WRITE_BAS_HEAD`

Controls whether the header section is written to the basic solution file.

Default
"ON"

Accepted
"ON", *"OFF"*

Example
`param = [ "MSK_IPAR_WRITE_BAS_HEAD", "MSK_ON" ]`

Groups
Data input/output, *Solution input/output*

`MSK_IPAR_WRITE_BAS_VARIABLES`

Controls whether the variables section is written to the basic solution file.

Default
"ON"

Accepted
"ON", *"OFF"*

Example
`param = [ "MSK_IPAR_WRITE_BAS_VARIABLES", "MSK_ON" ]`

Groups
Data input/output, *Solution input/output*

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]

Example
`param = [ "MSK_IPAR_WRITE_COMPRESSION", "9" ]`

Groups
Data input/output

MSK_IPAR_WRITE_FREE_CON

Controls whether the free constraints are written to the data file. Applies to MPS files.

Default
"ON"

Accepted
"ON", *"OFF"*

Example
`param = [ "MSK_IPAR_WRITE_FREE_CON", "MSK_ON" ]`

Groups
Data input/output

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"*

Example
`param = [ "MSK_IPAR_WRITE_GENERIC_NAMES", "MSK_OFF" ]`

Groups
Data input/output

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.

Example
`param = [ "MSK_IPAR_WRITE_IGNORE_INCOMPATIBLE_ITEMS", "MSK_OFF" ]`

Groups
Data input/output

`MSK_IPAR_WRITE_INT_CONSTRAINTS`

Controls whether the constraint section is written to the integer solution file.

Default
"ON"

Accepted
"ON", *"OFF"*

Example
`param = [ "MSK_IPAR_WRITE_INT_CONSTRAINTS", "MSK_ON" ]`

Groups
Data input/output, *Solution input/output*

`MSK_IPAR_WRITE_INT_HEAD`

Controls whether the header section is written to the integer solution file.

Default
"ON"

Accepted
"ON", *"OFF"*

Example
`param = [ "MSK_IPAR_WRITE_INT_HEAD", "MSK_ON" ]`

Groups
Data input/output, *Solution input/output*

`MSK_IPAR_WRITE_INT_VARIABLES`

Controls whether the variables section is written to the integer solution file.

Default
"ON"

Accepted
"ON", *"OFF"*

Example
`param = [ "MSK_IPAR_WRITE_INT_VARIABLES", "MSK_ON" ]`

Groups
Data input/output, *Solution input/output*

`MSK_IPAR_WRITE_JSON_INDENTATION`

When set, the JSON task and solution files are written with indentation for better readability.

Default
"OFF"

Accepted
"ON", *"OFF"*

Example
`param = [ "MSK_IPAR_WRITE_JSON_INDENTATION", "MSK_OFF" ]`

Groups
Data input/output

`MSK_IPAR_WRITE_LP_FULL_OBJ`

Write all variables, including the ones with 0-coefficients, in the objective.

Default
"ON"

Accepted
"ON", *"OFF"*

Example
`param = [ "MSK_IPAR_WRITE_LP_FULL_OBJ", "MSK_ON" ]`

Groups
Data input/output

MSK_IPAR_WRITE_LP_LINE_WIDTH

Maximum width of line in an LP file written by **MOSEK**.

Default
80

Accepted
[40; +inf]

Example
`param = [ "MSK_IPAR_WRITE_LP_LINE_WIDTH", "80" ]`

Groups
Data input/output

MSK_IPAR_WRITE_MPS_FORMAT

Controls in which format the MPS file is written.

Default
"FREE"

Accepted
"STRICT", *"RELAXED"*, *"FREE"*, *"CPLEX"*

Example
`param = [ "MSK_IPAR_WRITE_MPS_FORMAT", "MSK_MPS_FORMAT_FREE" ]`

Groups
Data input/output

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"*

Example
`param = [ "MSK_IPAR_WRITE_MPS_INT", "MSK_ON" ]`

Groups
Data input/output

MSK_IPAR_WRITE_SOL_BARVARIABLES

Controls whether the symmetric matrix variables section is written to the solution file.

Default
"ON"

Accepted
"ON", *"OFF"*

Example
`param = [ "MSK_IPAR_WRITE_SOL_BARVARIABLES", "MSK_ON" ]`

Groups
Data input/output, *Solution input/output*

MSK_IPAR_WRITE_SOL_CONSTRAINTS

Controls whether the constraint section is written to the solution file.

Default
"ON"

Accepted
"ON", *"OFF"*

Example
`param = [ "MSK_IPAR_WRITE_SOL_CONSTRAINTS", "MSK_ON" ]`

Groups
Data input/output, *Solution input/output*

MSK_IPAR_WRITE_SOL_HEAD

Controls whether the header section is written to the solution file.

Default
"ON"

Accepted
"ON", *"OFF"*

Example
`param = [ "MSK_IPAR_WRITE_SOL_HEAD", "MSK_ON" ]`

Groups
Data input/output, *Solution input/output*

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"*

Example
`param = [ "MSK_IPAR_WRITE_SOL_IGNORE_INVALID_NAMES", "MSK_OFF" ]`

Groups
Data input/output, *Solution input/output*

MSK_IPAR_WRITE_SOL_VARIABLES

Controls whether the variables section is written to the solution file.

Default
"ON"

Accepted
"ON", *"OFF"*

Example
`param = [ "MSK_IPAR_WRITE_SOL_VARIABLES", "MSK_ON" ]`

Groups
Data input/output, *Solution input/output*

13.5.3 String parameters

MSK_SPAR_BAS_SOL_FILE_NAME

Name of the `bas` solution file.

Accepted
Any valid file name.

Example
`param = [ "MSK_SPAR_BAS_SOL_FILE_NAME", "somevalue" ]`

Groups
Data input/output, *Solution input/output*

MSK_SPAR_DATA_FILE_NAME

Data are read and written to this file.

Accepted
Any valid file name.

Example
`param = [ "MSK_SPAR_DATA_FILE_NAME", "somevalue" ]`

Groups
: *Data input/output*

MSK_SPAR_DEBUG_FILE_NAME

MOSEK debug file.

Accepted
: Any valid file name.

Example
: `param = [ "MSK_SPAR_DEBUG_FILE_NAME", "somevalue" ]`

Groups
: *Data input/output*

MSK_SPAR_INT_SOL_FILE_NAME

Name of the `int` solution file.

Accepted
: Any valid file name.

Example
: `param = [ "MSK_SPAR_INT_SOL_FILE_NAME", "somevalue" ]`

Groups
: *Data input/output*, *Solution input/output*

MSK_SPAR_ITR_SOL_FILE_NAME

Name of the `itr` solution file.

Accepted
: Any valid file name.

Example
: `param = [ "MSK_SPAR_ITR_SOL_FILE_NAME", "somevalue" ]`

Groups
: *Data input/output*, *Solution input/output*

MSK_SPAR_MIO_DEBUG_STRING

For internal debugging purposes.

Accepted
: Any valid string.

Example
: `param = [ "MSK_SPAR_MIO_DEBUG_STRING", "somevalue" ]`

Groups
: *Data input/output*

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.

Example
: `param = [ "MSK_SPAR_PARAM_COMMENT_SIGN", "%%" ]`

Groups
: *Data input/output*

MSK_SPAR_PARAM_READ_FILE_NAME

Modifications to the parameter database is read from this file.

Accepted
Any valid file name.

Example
`param = [ "MSK_SPAR_PARAM_READ_FILE_NAME", "somevalue" ]`

Groups
Data input/output

MSK_SPAR_PARAM_WRITE_FILE_NAME

The parameter database is written to this file.

Accepted
Any valid file name.

Example
`param = [ "MSK_SPAR_PARAM_WRITE_FILE_NAME", "somevalue" ]`

Groups
Data input/output

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.

Example
`param = [ "MSK_SPAR_READ_MPS_BOU_NAME", "somevalue" ]`

Groups
Data input/output

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.

Example
`param = [ "MSK_SPAR_READ_MPS_OBJ_NAME", "somevalue" ]`

Groups
Data input/output

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.

Example
`param = [ "MSK_SPAR_READ_MPS_RAN_NAME", "somevalue" ]`

Groups
Data input/output

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.

Example
`param = [ "MSK_SPAR_READ_MPS_RHS_NAME", "somevalue" ]`

Groups
Data input/output

MSK_SPAR_REMOTE_OPTSERVER_HOST

URL of the remote optimization server in the format (http|https)://server:port. If set, all subsequent calls to any **MOSEK** function that involves synchronous optimization will be sent to the specified OptServer instead of being executed locally. Passing empty string deactivates this redirection.

Accepted
Any valid URL.

Example
`param = [ "MSK_SPAR_REMOTE_OPTSERVER_HOST", "somevalue" ]`

Groups
Overall system

MSK_SPAR_REMOTE_TLS_CERT

List of known server certificates in PEM format.

Accepted
PEM files separated by new-lines.

Example
`param = [ "MSK_SPAR_REMOTE_TLS_CERT", "somevalue" ]`

Groups
Overall system

MSK_SPAR_REMOTE_TLS_CERT_PATH

Path to known server certificates in PEM format.

Accepted
Any valid path.

Example
`param = [ "MSK_SPAR_REMOTE_TLS_CERT_PATH", "somevalue" ]`

Groups
Overall system

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.

Example
`param = [ "MSK_SPAR_SENSITIVITY_FILE_NAME", "somevalue" ]`

Groups
Data input/output

MSK_SPAR_SENSITIVITY_RES_FILE_NAME

Accepted
Any valid string.

Example
`param = [ "MSK_SPAR_SENSITIVITY_RES_FILE_NAME", "somevalue" ]`

Groups
Data input/output

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.

Example

```
param = [ "MSK_SPAR_SOL_FILTER_XC_LOW", "somevalue" ]
```

Groups

Data input/output, *Solution input/output*

`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.

Example

```
param = [ "MSK_SPAR_SOL_FILTER_XC_UPR", "somevalue" ]
```

Groups

Data input/output, *Solution input/output*

`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.

Example

```
param = [ "MSK_SPAR_SOL_FILTER_XX_LOW", "somevalue" ]
```

Groups

Data input/output, *Solution input/output*

`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.

Example

```
param = [ "MSK_SPAR_SOL_FILTER_XX_UPR", "somevalue" ]
```

Groups

Data input/output, *Solution input/output*

`MSK_SPAR_STAT_KEY`

Key used when writing the summary file.

Accepted

Any valid string.

Example

```
param = [ "MSK_SPAR_STAT_KEY", "somevalue" ]
```

Groups

Data input/output

`MSK_SPAR_STAT_NAME`

Name used when writing the statistics file.

Accepted

Any valid XML string.

Example

```
param = [ "MSK_SPAR_STAT_NAME", "somevalue" ]
```

Groups

Data input/output

13.6 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.

13.6.1 Termination

`"MSK_RES_OK" (0)`

No error occurred.

`"MSK_RES_TRM_MAX_ITERATIONS" (100000)`

The optimizer terminated at the maximum number of iterations.

`"MSK_RES_TRM_MAX_TIME" (100001)`

The optimizer terminated at the maximum amount of time.

`"MSK_RES_TRM_OBJECTIVE_RANGE" (100002)`

The optimizer terminated with an objective value outside the objective range.

`"MSK_RES_TRM_MIO_NUM_RELAXS" (100008)`

The mixed-integer optimizer terminated as the maximum number of relaxations was reached.

`"MSK_RES_TRM_MIO_NUM_BRANCHES" (100009)`

The mixed-integer optimizer terminated as the maximum number of branches was reached.

`"MSK_RES_TRM_NUM_MAX_NUM_INT_SOLUTIONS" (100015)`

The mixed-integer optimizer terminated as the maximum number of feasible solutions was reached.

`"MSK_RES_TRM_STALL" (100006)`

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" (100007)`

The optimizer terminated due to the return of the user-defined callback function.

`"MSK_RES_TRM_MAX_NUM_SETBACKS" (100020)`

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" (100025)`

The optimizer terminated due to numerical problems.

`"MSK_RES_TRM_LOST_RACE" (100027)`

Lost a race.

`"MSK_RES_TRM_INTERNAL" (100030)`

The optimizer terminated due to some internal reason. Please contact **MOSEK** support.

"MSK_RES_TRM_INTERNAL_STOP" (100031)

The optimizer terminated for internal reasons. Please contact **MOSEK** support.

"MSK_RES_TRM_SERVER_MAX_TIME" (100032)

remote server terminated **MOSEK** on time limit criteria.

"MSK_RES_TRM_SERVER_MAX_MEMORY" (100033)

remote server terminated **MOSEK** on memory limit criteria.

13.6.2 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 invalid. For instance when writing an LP file the names must not contain blanks or start with a digit. Also remeber to give the objective function a name.

"MSK_RES_WRN_INVALID_MPS_NAME" (504)
: A name e.g. a row name is not a valid MPS name.

"MSK_RES_WRN_INVALID_MPS_OBJ_NAME" (505)
: The objective name is not a valid MPS name.

"MSK_RES_WRN_LICENSE_FEATURE_EXPIRE" (509)
: 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_PRESOLVE_PRIMAL_PERTURBATIONS" (803)

The presolve perturbed the bounds of the primal problem. This is an indication that the problem is nearly infeasible.

"MSK_RES_WRN_WRITE_CHANGED_NAMES" (830)

Some names were changed because they were invalid for the output file format.

"MSK_RES_WRN_WRITE_DISCARDED_CFIX" (831)

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_NO_INFEASIBILITY_REPORT_WHEN_MATRIX_VARIABLES" (930)

An infeasibility report is not available when the problem contains matrix variables.

"MSK_RES_WRN_GETDUAL_IGNORES_INTEGRALITY" (940)

Dualizer ignores integer variables and disjunctive constraints.

"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.

"MSK_RES_WRN_MODIFIED_DOUBLE_PARAMETER" (970)

A double parameter related to solver tolerances has a non-default value.

"MSK_RES_WRN_LARGE_FIJ" (980)

A numerically large value is specified for an $f_{i,j}$ element in F. The parameter *MSK_DPAR_DATA_TOL_AIJ_LARGE* controls when an $f_{i,j}$ is considered large.

"MSK_RES_WRN_PTF_UNKNOWN_SECTION" (981)

Unexpected section in PTF file

13.6.3 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_LICENSE_OLD_SERVER_VERSION" (1003)

The version of the FlexLM license server is too old. You should upgrade the license server to one matching this version of **MOSEK**. It will support this and all older versions of **MOSEK**.

This error can appear if the client was updated to a new version which includes an upgrade of the licensing module, making it incompatible with a much older license server.

"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 has been reached unexpectedly.

"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_READ_GZIP" (1067)

Error encountered in GZIP stream.

"MSK_RES_ERR_READ_ZSTD" (1068)

Error encountered in ZSTD stream.

"MSK_RES_ERR_READ_ASYNC" (1069)

Error encountered in async stream.

"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_SPARSITY_SPECIFICATION" (1073)

The sparsity included an index that was out of bounds of the shape.

"MSK_RES_ERR_MISMATCHING_DIMENSION" (1074)

Mismatching dimensions specified in arguments

"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_DIMENSION_SPECIFICATION" (1082)

Invalid dimension specification

"MSK_RES_ERR_AXIS_NAME_SPECIFICATION" (1083)

Invalid axis names specification

"MSK_RES_ERR_READ_PREMATURE_EOF" (1089)

Encountered premature end-of-file in input stream.

"MSK_RES_ERR_READ_FORMAT" (1090)

The specified format cannot be read.

"MSK_RES_ERR_WRITE_LP_INVALID_VAR_NAMES" (1091)

Invalid variable name. Cannot write valid LP file.

"MSK_RES_ERR_WRITE_LP_DUPLICATE_VAR_NAMES" (1092)
  Duplicate variable names. Cannot write valid LP file.

"MSK_RES_ERR_WRITE_LP_INVALID_CON_NAMES" (1093)
  Invalid constraint name. Cannot write valid LP file.

"MSK_RES_ERR_WRITE_LP_DUPLICATE_CON_NAMES" (1094)
  Duplicate constraint names. Cannot write valid LP file.

"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_INVALID_CON_KEY" (1107)
  An invalid constraint key occurred in an MPS file.

"MSK_RES_ERR_MPS_INVALID_BOUND_KEY" (1108)
  An invalid bound key occurred in an MPS file.

"MSK_RES_ERR_MPS_INVALID_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_MPS_INVALID_KEY" (1129)

An invalid indicator key occurred in an MPS file.

"MSK_RES_ERR_MPS_INVALID_INDICATOR_CONSTRAINT" (1130)

An invalid indicator constraint is used. It must not be a ranged constraint.

"MSK_RES_ERR_MPS_INVALID_INDICATOR_VARIABLE" (1131)

An invalid indicator variable is specfied. It must be a binary variable.

"MSK_RES_ERR_MPS_INVALID_INDICATOR_VALUE" (1132)

An invalid indicator value is specfied. It must be either 0 or 1.

"MSK_RES_ERR_MPS_INVALID_INDICATOR_QUADRATIC_CONSTRAINT" (1133)

A quadratic constraint can be be an indicator constraint.

"MSK_RES_ERR_OPF_SYNTAX" (1134)

Syntax error in an OPF file

"MSK_RES_ERR_OPF_PREMATURE_EOF" (1136)

Premature end of file in an OPF file.

"MSK_RES_ERR_OPF_MISMATCHED_TAG" (1137)

Mismatched end-tag in OPF file

"MSK_RES_ERR_OPF_DUPLICATE_BOUND" (1138)

Either upper or lower bound was specified twice in OPF file

"MSK_RES_ERR_OPF_DUPLICATE_CONSTRAINT_NAME" (1139)

Duplicate constraint name in OPF File

"MSK_RES_ERR_OPF_INVALID_CONE_TYPE" (1140)

Invalid cone type in OPF File

"MSK_RES_ERR_OPF_INCORRECT_TAG_PARAM" (1141)

Invalid number of parameters in start-tag in OPF File

"MSK_RES_ERR_OPF_INVALID_TAG" (1142)

Invalid start-tag in OPF File

"MSK_RES_ERR_OPF_DUPLICATE_CONE_ENTRY" (1143)

Same variable appears in multiple cones in OPF File

"MSK_RES_ERR_OPF_TOO_LARGE" (1144)

The problem is too large to be correctly loaded

"MSK_RES_ERR_OPF_DUAL_INTEGER_SOLUTION" (1146)

Dual solution values are not allowed in OPF File

"MSK_RES_ERR_LP_EMPTY" (1151)

The problem cannot be written to an LP formatted file.

"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_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_LP_EXPECTED_NUMBER" (1158)
: Expected a number in LP file

"MSK_RES_ERR_READ_LP_MISSING_END_TAG" (1159)
: Syntax error in LP file. Possibly missing End tag.

"MSK_RES_ERR_LP_INDICATOR_VAR" (1160)
: An indicator variable was not declared binary

"MSK_RES_ERR_LP_EXPECTED_OBJECTIVE" (1161)
: Expected an objective section in LP file

"MSK_RES_ERR_LP_EXPECTED_CONSTRAINT_RELATION" (1162)
: Expected constraint relation

"MSK_RES_ERR_LP_AMBIGUOUS_CONSTRAINT_BOUND" (1163)
: Constraint has ambiguous or invalid bound

"MSK_RES_ERR_LP_DUPLICATE_SECTION" (1164)
: Duplicate section

"MSK_RES_ERR_READ_LP_DELAYED_ROWS_NOT_SUPPORTED" (1165)
: Duplicate section

"MSK_RES_ERR_WRITING_FILE" (1166)
: An error occurred while writing file

"MSK_RES_ERR_WRITE_ASYNC" (1167)
: An error occurred while performing asynchronous writing

"MSK_RES_ERR_INVALID_NAME_IN_SOL_FILE" (1170)
: An invalid name occurred in a solution 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_PTF_INCOMPATIBILITY" (1181)
: Incompatible item

"MSK_RES_ERR_PTF_UNDEFINED_ITEM" (1182)
: Undefined symbol referenced

"MSK_RES_ERR_PTF_INCONSISTENCY" (1183)
: Inconsistent size of item

"MSK_RES_ERR_PTF_FORMAT" (1184)
: Syntax error in an PTF 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_INDEX_IS_NOT_UNIQUE" (1205)
: An index in an argument is not unique.

"MSK_RES_ERR_PARAM_NAME" (1206)
: The parameter name is not correct.

"MSK_RES_ERR_PARAM_NAME_DOU" (1207)
: The parameter name is not correct for a double parameter.

"MSK_RES_ERR_PARAM_NAME_INT" (1208)
: The parameter name is not correct for an integer parameter.

"MSK_RES_ERR_PARAM_NAME_STR" (1209)
: 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_UNALLOWED_WHICHSOL" (1248)

The value od whichsol is not allowed.

"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_INV_RESCODE" (1261)

Invalid response code.

"MSK_RES_ERR_INV_IINF" (1262)

Invalid integer information item.

"MSK_RES_ERR_INV_LIINF" (1263)

Invalid long integer information item.

"MSK_RES_ERR_INV_DINF" (1264)

Invalid double information item.

"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.

"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.

"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.

"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.

"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) or is not finite.

"MSK_RES_ERR_UPPER_BOUND_IS_A_NAN" (1391)

The upper bound specified is not a number (nan) or is not finite.

"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_INF_IN_DOUBLE_DATA" (1451)
: An infinite 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` or `Inf`.

"MSK_RES_ERR_NAN_IN_BUC" (1462)
: u^c contains an invalid floating point value, i.e. a `NaN` of `Inf`.

"MSK_RES_ERR_INVALID_CFIX" (1469)
: An invalid fixed term in the objective is speficied.

"MSK_RES_ERR_NAN_IN_C" (1470)
: c contains an invalid floating point value, i.e. a `NaN` or `Inf`.

"MSK_RES_ERR_NAN_IN_BLX" (1471)
: l^x contains an invalid floating point value, i.e. a `NaN` or `Inf`.

"MSK_RES_ERR_NAN_IN_BUX" (1472)
: u^x contains an invalid floating point value, i.e. a `NaN` or `Inf`.

"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_INVALID_CJ" (1474)

c_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_READ_WRITE" (2563)

Failed to read or write due to an I/O error.

"MSK_RES_ERR_TASK_PREMATURE_EOF" (2564)

The Task file ended prematurely.

"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_INT64_TO_INT32_CAST" (3800)

A 64 bit integer could not be cast to a 32 bit integer.

"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 the simple cones (deprecated).

"MSK_RES_ERR_INVALID_FILE_FORMAT_FOR_QUADRATIC_TERMS" (4006)

The file format does not support a problem with quadratic terms.

"MSK_RES_ERR_INVALID_FILE_FORMAT_FOR_NONLINEAR" (4010)

The file format does not support a problem with nonlinear terms.

"MSK_RES_ERR_INVALID_FILE_FORMAT_FOR_DISJUNCTIVE_CONSTRAINTS" (4011)

The file format does not support a problem with disjunctive constraints.

"MSK_RES_ERR_INVALID_FILE_FORMAT_FOR_AFFINE_CONIC_CONSTRAINTS" (4012)

The file format does not support a problem with affine conic constraints.

"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_DUPLICATE_DOMAIN_NAMES" (4504)

Two domain names are identical.

"MSK_RES_ERR_DUPLICATE_DJC_NAMES" (4505)

Two disjunctive constraint names are identical.

"MSK_RES_ERR_NON_UNIQUE_ARRAY" (5000)

An array does not contain unique elements.

"MSK_RES_ERR_ARGUMENT_IS_TOO_SMALL" (5004)

The value of a function argument is too small.

"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" (7110)
: Duplicate VAR keyword.

"MSK_RES_ERR_CBF_DUPLICATE_INT" (7111)
: Duplicate INT keyword.

"MSK_RES_ERR_CBF_INVALID_VAR_TYPE" (7112)
: Invalid variable type.

"MSK_RES_ERR_CBF_INVALID_CON_TYPE" (7113)
: Invalid constraint type.

"MSK_RES_ERR_CBF_INVALID_DOMAIN_DIMENSION" (7114)
Invalid domain dimension.

"MSK_RES_ERR_CBF_DUPLICATE_OBJACOORD" (7115)
Duplicate index in OBJCOORD.

"MSK_RES_ERR_CBF_DUPLICATE_BCOORD" (7116)
Duplicate index in BCOORD.

"MSK_RES_ERR_CBF_DUPLICATE_ACOORD" (7117)
Duplicate index in ACOORD.

"MSK_RES_ERR_CBF_TOO_FEW_VARIABLES" (7118)
Too few variables defined.

"MSK_RES_ERR_CBF_TOO_FEW_CONSTRAINTS" (7119)
Too few constraints defined.

"MSK_RES_ERR_CBF_TOO_FEW_INTS" (7120)
Too few ints are specified.

"MSK_RES_ERR_CBF_TOO_MANY_INTS" (7121)
Too many ints are specified.

"MSK_RES_ERR_CBF_INVALID_INT_INDEX" (7122)
Invalid INT index.

"MSK_RES_ERR_CBF_UNSUPPORTED" (7123)
Unsupported feature is present.

"MSK_RES_ERR_CBF_DUPLICATE_PSDVAR" (7124)
Duplicate PSDVAR keyword.

"MSK_RES_ERR_CBF_INVALID_PSDVAR_DIMENSION" (7125)
Invalid PSDVAR dimension.

"MSK_RES_ERR_CBF_TOO_FEW_PSDVAR" (7126)
Too few variables defined.

"MSK_RES_ERR_CBF_INVALID_EXP_DIMENSION" (7127)
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" (7140)
Invalid number of cones.

"MSK_RES_ERR_CBF_INVALID_DIMENSION_OF_CONES" (7141)
Invalid number of cones.

"MSK_RES_ERR_CBF_INVALID_NUM_OBJACOORD" (7150)
: Invalid number of OBJACOORD.

"MSK_RES_ERR_CBF_INVALID_NUM_OBJFCOORD" (7151)
: Invalid number of OBJFCOORD.

"MSK_RES_ERR_CBF_INVALID_NUM_ACOORD" (7152)
: Invalid number of ACOORD.

"MSK_RES_ERR_CBF_INVALID_NUM_BCOORD" (7153)
: Invalid number of BCOORD.

"MSK_RES_ERR_CBF_INVALID_NUM_FCOORD" (7155)
: Invalid number of FCOORD.

"MSK_RES_ERR_CBF_INVALID_NUM_HCOORD" (7156)
: Invalid number of HCOORD.

"MSK_RES_ERR_CBF_INVALID_NUM_DCOORD" (7157)
: Invalid number of DCOORD.

"MSK_RES_ERR_CBF_EXPECTED_A_KEYWORD" (7158)
: Expected a key word.

"MSK_RES_ERR_CBF_INVALID_NUM_PSDCON" (7200)
: Invalid number of PSDCON.

"MSK_RES_ERR_CBF_DUPLICATE_PSDCON" (7201)
: Duplicate CON keyword.

"MSK_RES_ERR_CBF_INVALID_DIMENSION_OF_PSDCON" (7202)
: Invalid PSDCON dimension.

"MSK_RES_ERR_CBF_INVALID_PSDCON_INDEX" (7203)
: Invalid PSDCON index.

"MSK_RES_ERR_CBF_INVALID_PSDCON_VARIABLE_INDEX" (7204)
: Invalid PSDCON index.

"MSK_RES_ERR_CBF_INVALID_PSDCON_BLOCK_INDEX" (7205)
: Invalid PSDCON index.

"MSK_RES_ERR_CBF_UNSUPPORTED_CHANGE" (7210)
: The CHANGE section is not supported.

"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_MPS_WRITE_CPLEX_INVALID_CONE_TYPE" (7750)
: An invalid cone type occurs when writing a CPLEX formatted MPS file.

"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_GETDUAL_NOT_AVAILABLE" (7820)
: The simple dualizer is not available for this problem class.

"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

"MSK_RES_ERR_SERVER_ADDRESS" (8004)

Invalid address string

"MSK_RES_ERR_SERVER_CERTIFICATE" (8005)

Invalid TLS certificate format or path

"MSK_RES_ERR_SERVER_TLS_CLIENT" (8006)

Failed to create TLS cleint

"MSK_RES_ERR_SERVER_ACCESS_TOKEN" (8007)

Invalid access token

"MSK_RES_ERR_SERVER_PROBLEM_SIZE" (8008)

The size of the problem exceeds the dimensions permitted by the instance of the OptServer where it was run.

"MSK_RES_ERR_SERVER_HARD_TIMEOUT" (8009)

The hard timeout limit was reached on solver server, and the solver process was killed

"MSK_RES_ERR_DUPLICATE_INDEX_IN_A_SPARSE_MATRIX" (20050)

An element in a sparse matrix is specified twice.

"MSK_RES_ERR_DUPLICATE_INDEX_IN_AFEIDX_LIST" (20060)

An index is specified twice in an affine expression list.

"MSK_RES_ERR_DUPLICATE_FIJ" (20100)

An element in the F matrix is specified twice.

"MSK_RES_ERR_INVALID_FIJ" (20101)

$f_{i,j}$ contains an invalid floating point value, i.e. a `NaN` or an infinite value.

"MSK_RES_ERR_HUGE_FIJ" (20102)

A numerically huge value is specified for an $f_{i,j}$ element in F. The parameter *MSK_DPAR_DATA_TOL_AIJ_HUGE* controls when an $f_{i,j}$ is considered huge.

"MSK_RES_ERR_INVALID_G" (20103)

g contains an invalid floating point value, i.e. a `NaN` or an infinite value.

"MSK_RES_ERR_INVALID_B" (20150)

b contains an invalid floating point value, i.e. a `NaN` or an infinite value.

"MSK_RES_ERR_DOMAIN_INVALID_INDEX" (20400)

A domain index is invalid.

"MSK_RES_ERR_DOMAIN_DIMENSION" (20401)

A domain dimension is invalid.

"MSK_RES_ERR_DOMAIN_DIMENSION_PSD" (20402)

A PSD domain dimension is invalid.

"MSK_RES_ERR_NOT_POWER_DOMAIN" (20403)

The function is only applicable to primal and dual power cone domains.

"MSK_RES_ERR_DOMAIN_POWER_INVALID_ALPHA" (20404)

Alpha contains an invalid floating point value, i.e. a `NaN` or an infinite value.

"MSK_RES_ERR_DOMAIN_POWER_NEGATIVE_ALPHA" (20405)

Alpha contains a negative value or zero.

"MSK_RES_ERR_DOMAIN_POWER_NLEFT" (20406)

The value of n_{left} is not in $[1, n-1]$ where n is the dimension.

"MSK_RES_ERR_AFE_INVALID_INDEX" (20500)

An affine expression index is invalid.

"MSK_RES_ERR_ACC_INVALID_INDEX" (20600)

A affine conic constraint index is invalid.

"MSK_RES_ERR_ACC_INVALID_ENTRY_INDEX" (20601)

The index of an element in an affine conic constraint is invalid.

"MSK_RES_ERR_ACC_AFE_DOMAIN_MISMATCH" (20602)

There is a mismatch between between the number of affine expressions and total dimension of the domain(s).

"MSK_RES_ERR_DJC_INVALID_INDEX" (20700)

A disjunctive constraint index is invalid.

"MSK_RES_ERR_DJC_UNSUPPORTED_DOMAIN_TYPE" (20701)

An unsupported domain type has been used in a disjunctive constraint.

"MSK_RES_ERR_DJC_AFE_DOMAIN_MISMATCH" (20702)

There is a mismatch between the number of affine expressions and total dimension of the domain(s).

"MSK_RES_ERR_DJC_INVALID_TERM_SIZE" (20703)

A termize is invalid.

"MSK_RES_ERR_DJC_DOMAIN_TERMSIZE_MISMATCH" (20704)

There is a mismatch between the number of domains and the term sizes.

"MSK_RES_ERR_DJC_TOTAL_NUM_TERMS_MISMATCH" (20705)

There total number of terms in all domains does not match.

"MSK_RES_ERR_UNDEF_SOLUTION" (22000)

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_NO_DOTY" (22010)

No doty is available

13.7 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.

`simprecision`

Experimental. Usage not recommended.

"MSK_SIM_PRECISION_NORMAL"

Experimental. Usage not recommended.

"MSK_SIM_PRECISION_EXTENDED"

Experimental. Usage not recommended.

`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.

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_FOLDING"

The calback function is called at the beginning of folding.

"MSK_CALLBACK_BEGIN_FOLDING_BI"

TBD

"MSK_CALLBACK_BEGIN_FOLDING_BI_DUAL"

TBD

"MSK_CALLBACK_BEGIN_FOLDING_BI_INITIALIZE"

TBD

"MSK_CALLBACK_BEGIN_FOLDING_BI_OPTIMIZER"

TBD

"MSK_CALLBACK_BEGIN_FOLDING_BI_PRIMAL"

TBD

"MSK_CALLBACK_BEGIN_INFEAS_ANA"

The callback function is called when the infeasibility analyzer is started.

"MSK_CALLBACK_BEGIN_INITIALIZE_BI"

The callback function is called from within the basis identification procedure when the initialization phase 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_OPTIMIZE_BI"

TBD.

"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_SOLVE_ROOT_RELAX"

The callback function is called when solution of root relaxation 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_DECOMP_MIO"

The callback function is called when the dedicated algorithm for independent blocks inside the mixed-integer solver is started.

"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_FOLDING"

The calback function is called at the end of folding.

"MSK_CALLBACK_END_FOLDING_BI"

TBD

"MSK_CALLBACK_END_FOLDING_BI_DUAL"

TBD

"MSK_CALLBACK_END_FOLDING_BI_INITIALIZE"

TBD

"MSK_CALLBACK_END_FOLDING_BI_OPTIMIZER"

TBD

"MSK_CALLBACK_END_FOLDING_BI_PRIMAL"

TBD

"MSK_CALLBACK_END_INFEAS_ANA"

The callback function is called when the infeasibility analyzer is terminated.

"MSK_CALLBACK_END_INITIALIZE_BI"

The callback function is called from within the basis identification procedure when the initialization phase 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_OPTIMIZE_BI"

TBD.

"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_SOLVE_ROOT_RELAX"

The callback function is called when solution of root relaxation is terminated.

"MSK_CALLBACK_END_TO_CONIC"

End conic reformulation.

"MSK_CALLBACK_END_WRITE"

MOSEK has finished writing a problem file.

"MSK_CALLBACK_FOLDING_BI_DUAL"

TBD

"MSK_CALLBACK_FOLDING_BI_OPTIMIZER"

TBD

"MSK_CALLBACK_FOLDING_BI_PRIMAL"

TBD

"MSK_CALLBACK_HEARTBEAT"

A heartbeat callback.

"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_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_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_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_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_OPTIMIZE_BI"

TBD.

"MSK_CALLBACK_PRIMAL_SIMPLEX"

The callback function is called from within the primal simplex optimizer.

"MSK_CALLBACK_QO_REFORMULATE"

The callback function is called at an intermediate stage of the conic quadratic reformulation.

"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_RESTART_MIO"

The callback function is called when the mixed-integer optimizer is restarted.

"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_UPDATE_SIMPLEX"

The callback function is called from simplex optimizer.

"MSK_CALLBACK_WRITE_OPF"

The callback function is called from the OPF writer.

`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.

`domaintype`

Cone types

`"MSK_DOMAIN_R"`

R.

`"MSK_DOMAIN_RZERO"`

The zero vector.

`"MSK_DOMAIN_RPLUS"`

The positive orthant.

`"MSK_DOMAIN_RMINUS"`

The negative orthant.

`"MSK_DOMAIN_QUADRATIC_CONE"`

The quadratic cone.

`"MSK_DOMAIN_RQUADRATIC_CONE"`

The rotated quadratic cone.

`"MSK_DOMAIN_PRIMAL_EXP_CONE"`

The primal exponential cone.

`"MSK_DOMAIN_DUAL_EXP_CONE"`

The dual exponential cone.

`"MSK_DOMAIN_PRIMAL_POWER_CONE"`

The primal power cone.

"MSK_DOMAIN_DUAL_POWER_CONE"

The dual power cone.

"MSK_DOMAIN_PRIMAL_GEO_MEAN_CONE"

The primal geometric mean cone.

"MSK_DOMAIN_DUAL_GEO_MEAN_CONE"

The dual geometric mean cone.

"MSK_DOMAIN_SVEC_PSD_CONE"

The vectorized positive semidefinite 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.

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.

solformat

Data format types

"MSK_SOL_FORMAT_EXTENSION"

The file extension is used to determine the data file format.

"MSK_SOL_FORMAT_B"

Simple binary format

"MSK_SOL_FORMAT_TASK"

Tar based format.

"MSK_SOL_FORMAT_JSON_TASK"

JSON based format.

dinfitem

Double information items

"MSK_DINF_ANA_PRO_SCALARIZED_CONSTRAINT_MATRIX_DENSITY"

Density percentage of the scalarized constraint matrix.

"MSK_DINF_BI_CLEAN_TIME"

Time spent within the clean-up phase of the basis identification procedure since its invocation (in seconds).

"MSK_DINF_BI_DUAL_TIME"

Time spent within the dual phase basis identification procedure since its invocation (in seconds).

"MSK_DINF_BI_PRIMAL_TIME"

Time spent within the primal phase of the basis identification procedure since its invocation (in seconds).

"MSK_DINF_BI_TIME"

Time spent within the basis identification procedure since its invocation (in seconds).

"MSK_DINF_FOLDING_BI_OPTIMIZE_TIME"

TBD

"MSK_DINF_FOLDING_BI_UNFOLD_DUAL_TIME"

TBD

"MSK_DINF_FOLDING_BI_UNFOLD_INITIALIZE_TIME"

TBD

"MSK_DINF_FOLDING_BI_UNFOLD_PRIMAL_TIME"

TBD

"MSK_DINF_FOLDING_BI_UNFOLD_TIME"

TBD

"MSK_DINF_FOLDING_FACTOR"

Problem size after folding as a fraction of the original size.

"MSK_DINF_FOLDING_TIME"

Total time spent in folding for continuous problems (in seconds).

"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 (in seconds).

"MSK_DINF_MIO_CLIQUE_SELECTION_TIME"

Selection time for clique cuts (in seconds).

"MSK_DINF_MIO_CLIQUE_SEPARATION_TIME"

Separation time for clique cuts (in seconds).

"MSK_DINF_MIO_CMIR_SELECTION_TIME"

Selection time for CMIR cuts (in seconds).

"MSK_DINF_MIO_CMIR_SEPARATION_TIME"

Separation time for CMIR cuts (in seconds).

"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_SELECTION_TIME"

Selection time for GMI cuts (in seconds).

"MSK_DINF_MIO_GMI_SEPARATION_TIME"

Separation time for GMI cuts (in seconds).

"MSK_DINF_MIO_IMPLIED_BOUND_SELECTION_TIME"

Selection time for implied bound cuts (in seconds).

"MSK_DINF_MIO_IMPLIED_BOUND_SEPARATION_TIME"

Separation time for implied bound cuts (in seconds).

"MSK_DINF_MIO_INITIAL_FEASIBLE_SOLUTION_OBJ"

If the user provided solution was found to be feasible this information item contains it's objective value.

"MSK_DINF_MIO_KNAPSACK_COVER_SELECTION_TIME"

Selection time for knapsack cover (in seconds).

"MSK_DINF_MIO_KNAPSACK_COVER_SEPARATION_TIME"

Separation time for knapsack cover (in seconds).

"MSK_DINF_MIO_LIPRO_SELECTION_TIME"

Selection time for lift-and-project cuts (in seconds).

"MSK_DINF_MIO_LIPRO_SEPARATION_TIME"

Separation time for lift-and-project cuts (in seconds).

"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 (in seconds).

"MSK_DINF_MIO_ROOT_CUT_SELECTION_TIME"

Total time for cut selection (in seconds).

"MSK_DINF_MIO_ROOT_CUT_SEPARATION_TIME"

Total time for cut separation (in seconds).

"MSK_DINF_MIO_ROOT_OPTIMIZER_TIME"

Time spent in the contiuous optimizer while processing the root node relaxation (in seconds).

"MSK_DINF_MIO_ROOT_PRESOLVE_TIME"

Time spent presolving the problem at the root node (in seconds).

"MSK_DINF_MIO_ROOT_TIME"

Time spent processing the root node (in seconds).

"MSK_DINF_MIO_SYMMETRY_DETECTION_TIME"

Total time for symmetry detection (in seconds).

"MSK_DINF_MIO_SYMMETRY_FACTOR"

Degree to which the problem is affected by detected symmetry.

"MSK_DINF_MIO_TIME"

Time spent in the mixed-integer optimizer (in seconds).

"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_TICKS"

Total number of ticks spent in the optimizer since it was invoked. It is strictly negative if it is not available.

"MSK_DINF_OPTIMIZER_TIME"

Total time spent in the optimizer since it was invoked (in seconds).

"MSK_DINF_PRESOLVE_ELI_TIME"

Total time spent in the eliminator since the presolve was invoked (in seconds).

"MSK_DINF_PRESOLVE_LINDEP_TIME"

Total time spent in the linear dependency checker since the presolve was invoked (in seconds).

"MSK_DINF_PRESOLVE_TIME"

Total time spent in the presolve since it was invoked (in seconds).

"MSK_DINF_PRESOLVE_TOTAL_PRIMAL_PERTURBATION"

Total perturbation of the bounds of the primal problem.

"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 (in seconds).

"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_READ_DATA_TIME"

Time spent reading the data file (in seconds).

"MSK_DINF_REMOTE_TIME"

The total real time in seconds spent when optimizing on a server by the process performing the optimization on the server (in seconds).

"MSK_DINF_SIM_DUAL_TIME"

Time spent in the dual simplex optimizer since invoking it (in seconds).

"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 (in seconds).

"MSK_DINF_SIM_TIME"

Time spent in the simplex optimizer since invoking it (in seconds).

"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_PVIOLACC"

Maximal primal violation for affine conic constraints in 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_PVIOLDJC"

Maximal primal violation for disjunctive 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_DVIOLACC"

Maximal dual violation for the affine conic constraints in 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 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_PVIOLACC"

Maximal primal violation for affine conic constraints in 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 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 (in seconds).

"MSK_DINF_WRITE_DATA_TIME"

Time spent writing the data file (in seconds).

feature

License feature

"MSK_FEATURE_PTS"

Base system.

"MSK_FEATURE_PTON"

Conic extension.

liinfitem

Long integer information items.

"MSK_LIINF_ANA_PRO_SCALARIZED_CONSTRAINT_MATRIX_NUM_COLUMNS"

Number of columns in the scalarized constraint matrix.

"MSK_LIINF_ANA_PRO_SCALARIZED_CONSTRAINT_MATRIX_NUM_NZ"

Number of non-zero entries in the scalarized constraint matrix.

"MSK_LIINF_ANA_PRO_SCALARIZED_CONSTRAINT_MATRIX_NUM_ROWS"

Number of rows in the scalarized constraint matrix.

"MSK_LIINF_BI_CLEAN_ITER"

Number of 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_FOLDING_BI_DUAL_ITER"

TBD

"MSK_LIINF_FOLDING_BI_OPTIMIZER_ITER"

TBD

"MSK_LIINF_FOLDING_BI_PRIMAL_ITER"

TBD

"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 problem to be solved by the mixed-integer optimizer.

"MSK_LIINF_MIO_FINAL_ANZ"

Number of non-zero entries in the constraint matrix of the mixed-integer optimizer's final problem.

"MSK_LIINF_MIO_INTPNT_ITER"

Number of interior-point iterations performed by the mixed-integer optimizer.

"MSK_LIINF_MIO_NUM_DUAL_ILLPOSED_CER"

Number of dual illposed certificates encountered by the mixed-integer optimizer.

"MSK_LIINF_MIO_NUM_PRIM_ILLPOSED_CER"

Number of primal illposed certificates encountered 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_NUMACC"

Number of affince conic constraints.

"MSK_LIINF_RD_NUMANZ"

Number of non-zeros in A that is read.

"MSK_LIINF_RD_NUMDJC"

Number of disjuncive constraints.

"MSK_LIINF_RD_NUMQNZ"

Number of Q non-zeros.

"MSK_LIINF_SIMPLEX_ITER"

Number of iterations performed by the simplex optimizer.

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_FOLDING_APPLIED"

Non-zero if folding was exploited.

"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_FINAL_NUMBIN"

Number of binary variables in the mixed-integer optimizer's final problem.

"MSK_IINF_MIO_FINAL_NUMBINCONEVAR"

Number of binary cone variables in the mixed-integer optimizer's final problem.

"MSK_IINF_MIO_FINAL_NUMCON"

Number of constraints in the mixed-integer optimizer's final problem.

"MSK_IINF_MIO_FINAL_NUMCONE"

Number of cones in the mixed-integer optimizer's final problem.

"MSK_IINF_MIO_FINAL_NUMCONEVAR"

Number of cone variables in the mixed-integer optimizer's final problem.

"MSK_IINF_MIO_FINAL_NUMCONT"

Number of continuous variables in the mixed-integer optimizer's final problem.

"MSK_IINF_MIO_FINAL_NUMCONTCONEVAR"

Number of continuous cone variables in the mixed-integer optimizer's final problem.

"MSK_IINF_MIO_FINAL_NUMDEXPCONES"

Number of dual exponential cones in the mixed-integer optimizer's final problem.

"MSK_IINF_MIO_FINAL_NUMDJC"

Number of disjunctive constraints in the mixed-integer optimizer's final problem.

"MSK_IINF_MIO_FINAL_NUMDPOWCONES"

Number of dual power cones in the mixed-integer optimizer's final problem.

"MSK_IINF_MIO_FINAL_NUMINT"

Number of integer variables in the mixed-integer optimizer's final problem.

"MSK_IINF_MIO_FINAL_NUMINTCONEVAR"

Number of integer cone variables in the mixed-integer optimizer's final problem.

"MSK_IINF_MIO_FINAL_NUMPEXPCONES"

Number of primal exponential cones in the mixed-integer optimizer's final problem.

"MSK_IINF_MIO_FINAL_NUMPPOWCONES"

Number of primal power cones in the mixed-integer optimizer's final problem.

"MSK_IINF_MIO_FINAL_NUMQCONES"

Number of quadratic cones in the mixed-integer optimizer's final problem.

"MSK_IINF_MIO_FINAL_NUMRQCONES"

Number of rotated quadratic cones in the mixed-integer optimizer's final problem.

"MSK_IINF_MIO_FINAL_NUMVAR"

Number of variables in the mixed-integer optimizer's final problem.

"MSK_IINF_MIO_INITIAL_FEASIBLE_SOLUTION"

This item informs if **MOSEK** found the solution provided by the user to be feasible

- 0: solution provided by the user was not found to be feasible for the current problem,
- 1: user provided solution was found to be feasible.

"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_ACTIVE_ROOT_CUTS"

Number of active cuts in the final relaxation after the mixed-integer optimizer's root cut generation.

"MSK_IINF_MIO_NUM_BLOCKS_SOLVED_IN_BB"

Number of independent decomposition blocks solved though a dedicated algorithm.

"MSK_IINF_MIO_NUM_BLOCKS_SOLVED_IN_PRESOLVE"

Number of independent decomposition blocks solved during presolve.

"MSK_IINF_MIO_NUM_BRANCH"

Number of branches performed during the optimization.

"MSK_IINF_MIO_NUM_INT_SOLUTIONS"

Number of integer feasible solutions that have been found.

"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_NUM_RESTARTS"

Number of restarts performed during the optimization.

"MSK_IINF_MIO_NUM_ROOT_CUT_ROUNDS"

Number of cut separation rounds at the root node of the mixed-integer optimizer.

"MSK_IINF_MIO_NUM_SELECTED_CLIQUE_CUTS"

Number of clique cuts selected to be included in the relaxation.

"MSK_IINF_MIO_NUM_SELECTED_CMIR_CUTS"

Number of Complemented Mixed Integer Rounding (CMIR) cuts selected to be included in the relaxation.

"MSK_IINF_MIO_NUM_SELECTED_GOMORY_CUTS"

Number of Gomory cuts selected to be included in the relaxation.

"MSK_IINF_MIO_NUM_SELECTED_IMPLIED_BOUND_CUTS"

Number of implied bound cuts selected to be included in the relaxation.

"MSK_IINF_MIO_NUM_SELECTED_KNAPSACK_COVER_CUTS"

Number of clique cuts selected to be included in the relaxation.

"MSK_IINF_MIO_NUM_SELECTED_LIPRO_CUTS"

Number of lift-and-project cuts selected to be included in the relaxation.

"MSK_IINF_MIO_NUM_SEPARATED_CLIQUE_CUTS"

Number of separated clique cuts.

"MSK_IINF_MIO_NUM_SEPARATED_CMIR_CUTS"

Number of separated Complemented Mixed Integer Rounding (CMIR) cuts.

"MSK_IINF_MIO_NUM_SEPARATED_GOMORY_CUTS"

Number of separated Gomory cuts.

"MSK_IINF_MIO_NUM_SEPARATED_IMPLIED_BOUND_CUTS"

Number of separated implied bound cuts.

"MSK_IINF_MIO_NUM_SEPARATED_KNAPSACK_COVER_CUTS"

Number of separated clique cuts.

"MSK_IINF_MIO_NUM_SEPARATED_LIPRO_CUTS"

Number of separated lift-and-project cuts.

"MSK_IINF_MIO_NUM_SOLVED_NODES"

Number of branch and bounds nodes solved in the main branch and bound tree.

"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_NUMDJC"

Number of disjunctive constraints 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_NUMDJC"

Number of disjunctive constraints 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_SELECTED_CUTS"

Total number of cuts selected to be included in the relaxation by the mixed-integer optimizer.

"MSK_IINF_MIO_TOTAL_NUM_SEPARATED_CUTS"

Total number of cuts separated 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_PRESOLVE_NUM_PRIMAL_PERTURBATIONS"

Number perturbations to thhe bounds of the primal problem.

"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.

`miqcqoreformmethod`

Specifies the reformulation method for mixed-integer quadratic problems.

`"MSK_MIO_QCQO_REFORMULATION_METHOD_FREE"`

The mixed-integer optimizer decides which reformulation method to apply.

`"MSK_MIO_QCQO_REFORMULATION_METHOD_NONE"`

No reformulation method is applied.

`"MSK_MIO_QCQO_REFORMULATION_METHOD_LINEARIZATION"`

A reformulation via linearization is applied.

`"MSK_MIO_QCQO_REFORMULATION_METHOD_EIGEN_VAL_METHOD"`

The eigenvalue method is applied.

`"MSK_MIO_QCQO_REFORMULATION_METHOD_DIAG_SDP"`

A perturbation of matrix diagonals via the solution of SDPs is applied.

`"MSK_MIO_QCQO_REFORMULATION_METHOD_RELAX_SDP"`

A Reformulation based on the solution of an SDP-relaxation of the problem is applied.

`miodatapermmethod`

Specifies the problem data permutation method for mixed-integer problems.

`"MSK_MIO_DATA_PERMUTATION_METHOD_NONE"`

No problem data permutation is applied.

`"MSK_MIO_DATA_PERMUTATION_METHOD_CYCLIC_SHIFT"`

A random cyclic shift is applied to permute the problem data.

"MSK_MIO_DATA_PERMUTATION_METHOD_RANDOM"

A random permutation is applied to the problem data.

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.

miovarseltype

Mixed-integer variable selection types

"MSK_MIO_VAR_SELECTION_FREE"

The optimizer decides the variable selection strategy.

"MSK_MIO_VAR_SELECTION_PSEUDOCOST"

The optimizer employs pseudocost variable selection.

"MSK_MIO_VAR_SELECTION_STRONG"

The optimizer employs strong branching varfiable selection

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_NEW_DUAL_SIMPLEX"

The new dual simplex optimizer is used.

"MSK_OPTIMIZER_NEW_PRIMAL_SIMPLEX"

The new primal simplex optimizer is used. It is not recommended to use this option.

"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. Note using this value almost always leads to a significantly slow down.

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.

foldingmode

Method of folding (symmetry detection for continuous problems).

"MSK_FOLDING_MODE_OFF"

Disabled.

"MSK_FOLDING_MODE_FREE"

The solver decides on the usage and amount of folding.

"MSK_FOLDING_MODE_FREE_UNLESS_BASIC"

If only the interior-point solution is requested then the solver decides; if the basic solution is requested then folding is disabled.

"MSK_FOLDING_MODE_FORCE"

Full folding is always performed regardless of workload.

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

"UNKNOWN"

Unknown problem status.

"PRIM_AND_DUAL_FEAS"

The problem is primal and dual feasible.

"PRIM_FEAS"

The problem is primal feasible.

"DUAL_FEAS"

The problem is dual feasible.

"PRIM_INFEAS"

The problem is primal infeasible.

"DUAL_INFEAS"

The problem is dual infeasible.

"PRIM_AND_DUAL_INFEAS"

The problem is primal and dual infeasible.

"ILL_POSED"

The problem is ill-posed. For example, it may be primal and dual feasible but have a positive duality gap.

"PRIM_INFEAS_OR_UNBOUNDED"

The problem is either primal infeasible or unbounded. This may occur for mixed-integer problems.

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.

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

`"UNKNOWN"`

Status of the solution is unknown.

`"OPTIMAL"`

The solution is optimal.

`"PRIM_FEAS"`

The solution is primal feasible.

`"DUAL_FEAS"`

The solution is dual feasible.

`"PRIM_AND_DUAL_FEAS"`

The solution is both primal and dual feasible.

`"PRIM_INFEAS_CER"`

The solution is a certificate of primal infeasibility.

`"DUAL_INFEAS_CER"`

The solution is a certificate of dual infeasibility.

`"PRIM_ILLPOSED_CER"`

The solution is a certificate that the primal problem is illposed.

`"DUAL_ILLPOSED_CER"`

The solution is a certificate that the dual problem is illposed.

`"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

`"UNK"`

The status for the constraint or variable is unknown.

`"BAS"`

The constraint or variable is in the basis.

`"SUPBAS"`

The constraint or variable is super basic.

"LOW"

The constraint or variable is at its lower bound.

"UPR"

The constraint or variable is at its upper bound.

"FIX"

The constraint or variable is fixed.

"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.

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.

13.8 Supported domains

This section lists the domains supported by **MOSEK**.

13.8.1 Linear domains

Each linear domain is determined by the dimension n.

- *"zero"* : the **zero domain**, consisting of the origin $0^n \in \mathbb{R}^n$.
- *"nonnegative"* : the **nonnegative orthant domain** $\mathbb{R}^n_{\geq 0}$.
- *"nonpositive"* : the **nonpositive orthant domain** $\mathbb{R}^n_{\leq 0}$.
- *"unbounded"* : the **free domain**, consisting of the whole $\mathbb{R}^n$.

Membership in a linear domain is equivalent to imposing the corresponding set of n linear constraints, for instance $Fx + g \in 0^n$ is equivalent to $Fx + g = 0$ and so on. The free domain imposes no restriction.

13.8.2 Quadratic cone domains

The quadratic domains are determined by the dimension n.

- *"quad"* : the **quadratic cone domain** is the subset of $\mathbb{R}^n$ defined as

$$\mathcal{Q}^n = \left\{ x \in \mathbb{R}^n \ : \ x_1 \geq \sqrt{x_2^2 + \cdots + x_n^2} \right\}.$$

- *"rquad"* : the **rotated quadratic cone domain** is the subset of $\mathbb{R}^n$ defined as

$$\mathcal{Q}_r^n = \left\{ x \in \mathbb{R}^n \ : \ 2x_1x_2 \geq x_3^2 + \cdots + x_n^2, \ x_1, x_2 \geq 0 \right\}.$$

13.8.3 Exponential cone domains

- *"exp"* : the **primal exponential cone domain** is the subset of $\mathbb{R}^3$ defined as

$$K_{\exp} = \left\{ (x_1, x_2, x_3) \in \mathbb{R}^3 \ : \ x_1 \geq x_2 \exp(x_3/x_2), \ x_1, x_2 \geq 0 \right\}.$$

- *"dexp"* : the **dual exponential cone domain** is the subset of $\mathbb{R}^3$ defined as

$$K_{\exp}^* = \left\{ (x_1, x_2, x_3) \in \mathbb{R}^3 \ : \ x_1 \geq -x_3 \exp(x_2/x_3 - 1), \ x_1 \geq 0, x_3 \leq 0 \right\}.$$

13.8.4 Power cone domains

A power cone domain is determined by the dimension n and a sequence of $1 \leq n_l < n$ positive real numbers (weights) $\alpha_1, \ldots, \alpha_{n_l}$.

- *"pow"* : the **primal power cone domain** is the subset of $\mathbb{R}^n$ defined as

$$\mathcal{P}_n^{(\alpha_1,\ldots,\alpha_{n_l})} = \left\{ x \in \mathbb{R}^n \ : \ \prod_{i=1}^{n_l} x_i^{\beta_i} \geq \sqrt{x_{n_l+1}^2 + \cdots + x_n^2}, \ x_1, \ldots, x_{n_l} \geq 0 \right\}.$$

  where β_i are the weights normalized to add up to 1, ie. $\beta_i = \alpha_i / (\sum_j \alpha_j)$ for $i = 1, \ldots, n_l$. The name n_l reads as “n left”, the length of the product on the left-hand side of the definition.

- *"dpow"* : the **dual power cone domain** is the subset of $\mathbb{R}^n$ defined as

$$\left(\mathcal{P}_n^{(\alpha_1,\ldots,\alpha_{n_l})}\right)^* = \left\{ x \in \mathbb{R}^n \ : \ \prod_{i=1}^{n_l} \left(\frac{x_i}{\beta_i}\right)^{\beta_i} \geq \sqrt{x_{n_l+1}^2 + \cdots + x_n^2}, \ x_1, \ldots, x_{n_l} \geq 0 \right\}.$$

  where β_i are the weights normalized to add up to 1, ie. $\beta_i = \alpha_i / (\sum_j \alpha_j)$ for $i = 1, \ldots, n_l$. The name n_l reads as “n left”, the length of the product on the left-hand side of the definition.

- **Remark:** in MOSEK 9 power cones were available only in the special case with $n_l = 2$ and weights $(\alpha, 1 - \alpha)$ for some $0 < \alpha < 1$ specified as cone parameter.

13.8.5 Geometric mean cone domains

A geometric mean cone domain is determined by the dimension n.

- *"geomean"* : the **primal geometric mean cone domain** is the subset of $\mathbb{R}^n$ defined as

  $$\mathcal{GM}^n = \left\{ x \in \mathbb{R}^n \ : \ \left(\prod_{i=1}^{n-1} x_i\right)^{1/(n-1)} \geq |x_n|, \ x_1, \ldots, x_{n-1} \geq 0 \right\}.$$

  It is a special case of the primal power cone domain with $n_l = n-1$ and weights $\alpha = (1, \ldots, 1)$.

- *"dgeomean"* : the **dual geometric mean cone domain** is the subset of $\mathbb{R}^n$ defined as

  $$(\mathcal{GM}^n)^* = \left\{ x \in \mathbb{R}^n \ : \ (n-1)\left(\prod_{i=1}^{n-1} x_i\right)^{1/(n-1)} \geq |x_n|, \ x_1, \ldots, x_{n-1} \geq 0 \right\}.$$

  It is a special case of the dual power cone domain with $n_l = n-1$ and weights $\alpha = (1, \ldots, 1)$.

13.9 Environment variables

This section lists operating system environment variables which can globally affect the behavior of **MOSEK**.

It is recommended that any environment variables are set in the environment *before* launching a process using **MOSEK**, or at the very least before the first time any **MOSEK** library or package is loaded by the process.

- `MOSEKLM_LICENSE_FILE` - location of license file.

  Commonly used to point **MOSEK** to a floating license token server or change the default license file search path. For details see the licensing guide.

- `MOSEK_SYS_NUM_CORES` - set the number of cores.

  When **MOSEK** is loaded it detects the number of cores available on the machine. Setting this environment variable overrides that detection.

  Most users will never need it. Typical applications would be:

  - When **MOSEK** fails to detect the number of cores correctly.
  - To limit the default number of cores available to **MOSEK** in a way transparent to the users.

- `PATH` - system search path.

  In all automated cases (MSI, package managers) the installation process will ensure that the **MOSEK** binaries can be located on runtime, either by adding them to the system search path or via other mechanisms.

  It may be needed to set up by hand for manual, modified or other custom installations.

- `LD_LIBRARY_PATH`, `DYLD_LIBRARY_PATH` - shared objects search path.

  Affects the locations where loader looks for shared libraries. Should never be needed for a correct installation.

- `MIMALLOC_PURGE_DELAY` - mimalloc page release delay.

  Setting it to `0` gives the most aggressive memory release behavior at the cost of speed.

  Most users should never change it. Setting `0` may decrease memory consumption in some special scenarios. Known cases include optimizing very large models many times in a row in the same process. Do not consider it unless memory use becomes an actual issue. Usage at own risk.

Chapter 14

Supported File Formats

MOSEK supports a range of problem and solution formats listed in Table 14.1 and Table 14.2.

The most important are:

- the **Task format**, **MOSEK**'s native binary format which supports all features that **MOSEK** supports. It is the closest possible representation of the internal data in a task and it is ideal for submitting problem data support questions.
- the **PTF format**, **MOSEK**'s human-readable format that supports all linear, conic and mixed-integer features. It is ideal for debugging. It is not an exact copy of all the data in the task, but it contains all information required to reconstruct it, presented in a readable fashion.
- **MPS**, **LP**, **CBF** formats are industry standards, each supporting some limited set of features, and potentially requiring some degree of reformulation during read/write.

Problem formats

Table 14.1: List of supported file formats for optimization problems.

Format Type	Ext.	Binary/Text	LP	QCQO	ACC	SDP	DJC	Sol	Param
LP	lp	plain text	X	X					
MPS	mps	plain text	X	X					
PTF	ptf	plain text	X		X	X	X	X	X
CBF	cbf	plain text	X		X	X			
Task format	task	binary	X	X	X	X	X	X	X
Jtask format	jtask	text/JSON	X	X	X	X	X	X	X
OPF (deprecated for conic problems)	opf	plain text	X	X				X	X

The columns of the table indicate if the specified file format supports:

- LP - linear problems, possibly with integer variables,
- QCQO - quadratic objective or constraints,
- ACC - affine conic constraints,
- SDP - semidefinite cone/variables,
- DJC - disjunctive constraints,
- Sol - solutions,
- Param - optimizer parameters.

Solution formats

Table 14.2: 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/JSON	All solutions

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.zst
```

will be considered as a Zstandard compressed MPS file.

14.1 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.

14.1.1 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.

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.

Ranged constraints cannot be written in LP format, and have to be split into a separate upper and lower bound.

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
```

(continues on next page)

(continued from previous page)

```
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
```

14.1.2 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
```

14.1.3 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.

14.2 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].

14.2.1 MPS File Structure

The version of the MPS format supported by **MOSEK** allows specification of an optimization problem of the form

$$\begin{array}{lccccl} \text{maximize/minimize} & & c^T x + q_0(x) & & & \\ l^c & \leq & Ax + q(x) & \leq & u^c, & \\ l^x & \leq & x & \leq & u^x, & \\ & & x \in \mathcal{K}, & & & \\ & & x_{\mathcal{J}} \ \ \text{integer}, & & & \end{array} \tag{14.1}$$

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]
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. 14.2.5 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
    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

$$\begin{array}{lll} \text{minimize} & -x_2 + \frac{1}{2}(2x_1^2 - 2x_1x_3 + 0.2x_2^2 + 2x_3^2) & \\ \text{subject to} & x_1 + x_2 + x_3 & \geq \quad 1, \\ & x \geq 0 & \end{array}$$

has the following MPS file representation

```
* File: qo1.mps
NAME          qo1
ROWS
 N  obj
 G  c1
COLUMNS
    x1        c1        1.0
    x2        obj       -1.0
    x2        c1        1.0
    x3        c1        1.0
RHS
    rhs       c1        1.0
QSECTION      obj
    x1        x1        2.0
    x1        x3        -1.0
    x2        x2        0.2
    x3        x3        2.0
ENDATA
```

Regarding the QSECTIONs please note that:

- Only one QSECTION is allowed for each constraint.
- The QSECTIONs can appear in an arbitrary order after the COLUMNS section.
- All variable names occurring in the QSECTION must already be specified in the COLUMNS section.
- All entries specified in a QSECTION are assumed to belong to the lower triangular part of the quadratic term of Q .

QMATRIX/QUADOBJ (optional)

The QMATRIX and QUADOBJ sections allow to define the quadratic term of the objective function. They differ in how the quadratic term of the objective function is stored:

- QMATRIX 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
    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 QCMATRIXs please note that:

- Only one QCMATRIX is allowed for each constraint.
- The QCMATRIXs can appear in an arbitrary order after the COLUMNS section.
- All variable names occurring in the QSECTION must already be specified in the COLUMNS section.
- 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 (14.1). 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:

$$\mathcal{K}_t = \{0\} \subseteq \mathbb{R}^{n^t}. \tag{14.2}$$

- Quadratic cone:

$$\mathcal{K}_t = \left\{ x \in \mathbb{R}^{n^t} : x_1 \geq \sqrt{\sum_{j=2}^{n^t} x_j^2} \right\}. \tag{14.3}$$

- Rotated quadratic cone:

$$\mathcal{K}_t = \left\{ x \in \mathbb{R}^{n^t} : 2x_1x_2 \geq \sum_{j=3}^{n^t} x_j^2, \quad x_1, x_2 \geq 0 \right\}. \tag{14.4}$$

- Primal exponential cone:

$$\mathcal{K}_t = \left\{ x \in \mathbb{R}^3 : x_1 \geq x_2 \exp(x_3/x_2), \quad x_1, x_2 \geq 0 \right\}. \tag{14.5}$$

- Primal power cone (with parameter $0 < \alpha < 1$):

$$\mathcal{K}_t = \left\{ x \in \mathbb{R}^{n^t} : x_1^{\alpha} x_2^{1-\alpha} \geq \sqrt{\sum_{j=3}^{n^t} x_j^2}, \quad x_1, x_2 \geq 0 \right\}. \tag{14.6}$$

- Dual exponential cone:

$$\mathcal{K}_t = \left\{ x \in \mathbb{R}^3 : x_1 \geq -x_3 e^{-1} \exp(x_2/x_3), \quad x_3 \leq 0, x_1 \geq 0 \right\}. \tag{14.7}$$

- Dual power cone (with parameter $0 < \alpha < 1$):

$$\mathcal{K}_t = \left\{ x \in \mathbb{R}^{n^t} : \left(\frac{x_1}{\alpha}\right)^{\alpha} \left(\frac{x_2}{1-\alpha}\right)^{1-\alpha} \geq \sqrt{\sum_{j=3}^{n^t} x_j^2}, \quad x_1, x_2 \geq 0 \right\}. \tag{14.8}$$

In general, membership in the $\mathbb{R}$ set is not specified. If a variable is not a member of any other cone then it is assumed to be a member of the $\mathbb{R}$ cone.

Next, let us study an example. Assume that the power cone

$$x_4^{1/3} x_5^{2/3} \geq |x_8|$$

and the rotated quadratic cone

$$2x_3x_7 \geq x_1^2 + x_0^2, \quad x_3, x_7 \geq 0,$$

should be specified in the MPS file. One `CSECTION` is required for each cone and they are specified as follows:

```
*       1         2         3         4         5         6
*234567890123456789012345678901234567890123456789012345678901234567890
CSECTION      konea     3e-1           PPOW
x4
x5
x8
CSECTION      koneb     0.0            RQUAD
x7
x3
x1
x0
```

In general, a `CSECTION` header has the format

```
CSECTION      [kname1]    [value1]     [ktype]
```

where the requirements for each field are as follows:

Field	Starting Position	Max Width	Required	Description
`[kname1]`	15	8	Yes	Name of the cone
`[value1]`	25	12	No	Cone parameter
`[ktype]`	40		Yes	Type of the cone.

The possible cone type keys are:

[ktype]	Members	[value1]	Interpretation.
ZERO	≥ 0	unused	Zero cone (14.2).
QUAD	≥ 1	unused	Quadratic cone (14.3).
RQUAD	≥ 2	unused	Rotated quadratic cone (14.4).
PEXP	3	unused	Primal exponential cone (14.5).
PPOW	≥ 2	α	Primal power cone (14.6).
DEXP	3	unused	Dual exponential cone (14.7).
DPOW	≥ 2	α	Dual power cone (14.8).

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.

14.2.2 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.

14.2.3 General Limitations

- An MPS file should be an ASCII file.

14.2.4 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.

14.2.5 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.

> **Warning:** This file format is to a large extent deprecated. While it can still be used for linear and quadratic problems, for conic problems the Sec. 14.5 is recommended.

14.3 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.

14.3.1 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.
```

(continues on next page)

(continued from previous page)

```
[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]
```

14.3.2 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

$$2x_1 x_2 \geq \sum_{i=3}^{n} x_i^2, \quad x_1, x_2 \geq 0.$$

- `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 $x_1, \ldots, x_n$ defines a constraint of the form

$$\left(\frac{x_1}{\alpha}\right)^{\alpha} \left(\frac{x_2}{1-\alpha}\right)^{1-\alpha} \geq \sqrt{\sum_{j=3}^{n^t} x_j^2}, \quad x_1, x_2 \geq 0.$$

- zero: zero cone of n variables $x_1, \ldots, x_n$ defines a constraint of the form

$$x_1 = \cdots = x_n = 0$$

A [bounds]-section example:

```
[bounds]
[b]  0 <= x,y <= 10  [/b] # ranged bound
[b] 10 >= x,y >=  0  [/b] # ranged bound
[b]  0 <= x,y <= inf [/b] # using inf
[b]       x,y free   [/b] # free variables
# Let (x,y,z,w) belong to the cone K
[cone rquad] x,y,z,w  [/cone] # rotated quadratic cone
[cone ppow '3e-01' 'a'] x1, x2, x3 [/cone] # power cone with alpha=1/3 and name 'a'
[/bounds]
```

By default all variables are free.

[variables]

This defines an ordering of variables as they should appear in the problem. This is simply a space-separated list of variable names.

[integer]

This contains a space-separated list of variables and defines the constraint that the listed variables must be integer-valued.

[hints]

This may contain only non-essential data; for example estimates of the number of variables, constraints and non-zeros. Placed before all other sections containing data this may reduce the time spent reading the file.

In the hints section, any subsection which is not recognized by **MOSEK** is simply ignored. In this section a hint is defined as follows:

```
[hint ITEM] value [/hint]
```

The hints recognized by **MOSEK** are:

- numvar (number of variables),
- numcon (number of linear/quadratic constraints),
- numanz (number of linear non-zeros in constraints),
- numqnz (number of quadratic non-zeros in constraints).

[solutions]

This section can contain a set of full or partial solutions to a problem. Each solution must be specified using a [solution]-section, i.e.

```
[solutions]
[solution]...[/solution] #solution 1
[solution]...[/solution] #solution 2
#other solutions....
[solution]...[/solution] #solution n
[/solutions]
```

The syntax of a [solution]-section is the following:

```
[solution SOLTYPE status=STATUS]...[/solution]
```

where SOLTYPE is one of the strings

- interior, a non-basic solution,
- basic, a basic solution,
- integer, an integer solution,

and STATUS is one of the strings

- UNKNOWN,
- OPTIMAL,
- INTEGER_OPTIMAL,
- PRIM_FEAS,
- DUAL_FEAS,
- PRIM_AND_DUAL_FEAS,
- NEAR_OPTIMAL,
- NEAR_PRIM_FEAS,
- NEAR_DUAL_FEAS,
- NEAR_PRIM_AND_DUAL_FEAS,
- PRIM_INFEAS_CER,
- DUAL_INFEAS_CER,
- NEAR_PRIM_INFEAS_CER,
- NEAR_DUAL_INFEAS_CER,
- NEAR_INTEGER_OPTIMAL.

Most of these values are irrelevant for input solutions; when constructing a solution for simplex hot-start or an initial solution for a mixed integer problem the safe setting is UNKNOWN.

A [solution]-section contains [con] and [var] sections. Each [con] and [var] section defines solution information for a single variable or constraint, specified as list of KEYWORD/value pairs, in any order, written as

```
KEYWORD=value
```

Allowed keywords are as follows:

- sk. The status of the item, where the value is one of the following strings:
  - LOW, the item is on its lower bound.

  - UPR, the item is on its upper bound.
  - FIX, it is a fixed item.
  - BAS, the item is in the basis.
  - SUPBAS, the item is super basic.
  - UNK, the status is unknown.
  - INF, the item is outside its bounds (infeasible).

- lvl Defines the level of the item.
- sl Defines the level of the dual variable associated with its lower bound.
- su Defines the level of the dual variable associated with its upper bound.
- sn Defines the level of the variable associated with its cone.
- y Defines the level of the corresponding dual variable (for constraints only).

A [var] section should always contain the items sk, lvl, sl and su. Items sl and su are not required for integer solutions.

A [con] section should always contain sk, lvl, sl, su and y.

An example of a solution section

```
[solution basic status=UNKNOWN]
[var x0] sk=LOW    lvl=5.0      [/var]
[var x1] sk=UPR    lvl=10.0     [/var]
[var x2] sk=SUPBAS lvl=2.0  sl=1.5 su=0.0 [/var]

[con c0] sk=LOW    lvl=3.0 y=0.0 [/con]
[con c0] sk=UPR    lvl=0.0 y=5.0 [/con]
[/solution]
```

- [vendor] This contains solver/vendor specific data. It accepts one argument, which is a vendor ID – for **MOSEK** the ID is simply mosek – and the section contains the subsection parameters defining solver parameters. When reading a vendor section, any unknown vendor can be safely ignored. This is described later.

Comments using the # may appear anywhere in the file. Between the # and the following line-break any text may be written, including markup characters.

14.3.3 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]+)?
```

14.3.4 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"
```

14.3.5 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]
```

14.3.6 Writing OPF Files from MOSEK

14.3.7 Examples

This section contains a set of small examples written in OPF and describing how to formulate linear, quadratic and conic problems.

Linear Example `lo1.opf`

Consider the example:

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

having the bounds

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

In the `OPF` format the example is displayed as shown in Listing 14.1.

Listing 14.1: Example of an OPF file for a linear problem.

```
[comment]
  The lo1 example in OPF format
[/comment]

[hints]
  [hint NUMVAR] 4 [/hint]
  [hint NUMCON] 3 [/hint]
  [hint NUMANZ] 9 [/hint]
[/hints]

[variables disallow_new_variables]
  x1 x2 x3 x4
[/variables]

[objective maximize 'obj']
   3 x1 + x2 + 5 x3 + x4
[/objective]

[constraints]
  [con 'c1'] 3 x1 +   x2 + 2 x3         = 30 [/con]
  [con 'c2'] 2 x1 +   x2 + 3 x3 +   x4 >= 15 [/con]
  [con 'c3']        2 x2        + 3 x4 <= 25 [/con]
[/constraints]

[bounds]
  [b] 0 <= * [/b]
  [b] 0 <= x2 <= 10 [/b]
[/bounds]
```

Quadratic Example `qo1.opf`

An example of a quadratic optimization problem is

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

This can be formulated in `opf` as shown below.

Listing 14.2: 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]
```

(continues on next page)

(continued from previous page)

```
[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:

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

Please note that the type of the cones is defined by the parameter to `[cone ...]`; the content of the `cone`-section is the names of variables that belong to the cone. The resulting OPF file is in Listing 14.3.

Listing 14.3: 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
```

(continues on next page)

(continued from previous page)

```
  [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 14.4.

Listing 14.4: Example of an OPF file for a mixed-integer linear problem.

```
[comment]
  The milo1 example in OPF format
[/comment]

[hints]
  [hint NUMVAR] 2 [/hint]
  [hint NUMCON] 2 [/hint]
  [hint NUMANZ] 4 [/hint]
[/hints]

[variables disallow_new_variables]
  x1 x2
[/variables]

[objective maximize 'obj']
   x1 + 6.4e-1 x2
[/objective]

[constraints]
  [con 'c1'] 5e+1 x1 + 3.1e+1 x2 <= 2.5e+2 [/con]
  [con 'c2'] -4 <= 3 x1 - 2 x2 [/con]
[/constraints]

[bounds]
  [b] 0 <= * [/b]
[/bounds]

[integer]
  x1 x2
[/integer]
```

14.4 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), exponential cone, power cone 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 CBF format separates problem structure from the problem data.

14.4.1 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:

$$\begin{array}{lll} \min / \max & g^{obj} & \\ & g_i \in \mathcal{K}_i, & i \in \mathcal{I}, \\ \text{s.t.} & G_i \in \mathcal{K}_i, & i \in \mathcal{I}^{PSD}, \\ & x_j \in \mathcal{K}_j, & j \in \mathcal{J}, \\ & \overline{X}_j \in \mathcal{K}_j, & j \in \mathcal{J}^{PSD}. \end{array} \tag{14.9}$$

- **Variables** are either scalar variables, x_j for $j \in \mathcal{J}$, or matrix 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}

$$g^{obj} = \sum_{j \in \mathcal{J}^{PSD}} \langle F_j^{obj}, X_j \rangle + \sum_{j \in \mathcal{J}} a_j^{obj} x_j + b^{obj}.$$

As of version 4 of the format, CBF files can represent the following non-parametric 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

$$\left\{ \begin{pmatrix} p \\ q \\ x \end{pmatrix} \in \mathbb{R} \times \mathbb{R} \times \mathbb{R}^{n-2},\ 2pq \geq x^T x,\ p \geq 0,\ q \geq 0 \right\},\ \text{for } n \geq 3.$$

- **Exponential cone** - A cone in the exponential cone family defined by

$$\text{cl}(S_1) = S_1 \cup S_2$$

  where,

$$S_1 = \left\{ \begin{pmatrix} t \\ s \\ r \end{pmatrix} \in \mathbb{R}^3,\ t \geq s e^{\frac{r}{s}},\ s \geq 0 \right\}.$$

  and,

$$S_2 = \left\{ \begin{pmatrix} t \\ s \\ r \end{pmatrix} \in \mathbb{R}^3,\ t \geq 0,\ r \leq 0,\ s = 0 \right\}.$$

- **Dual Exponential cone** - A cone in the exponential cone family defined by

$$\text{cl}(S_1) = S_1 \cup S_2$$

  where,

$$S_1 = \left\{ \begin{pmatrix} t \\ s \\ r \end{pmatrix} \in \mathbb{R}^3,\ et \geq (-r) e^{\frac{s}{r}},\ -r \geq 0 \right\}.$$

  and,

$$S_2 = \left\{ \begin{pmatrix} t \\ s \\ r \end{pmatrix} \in \mathbb{R}^3,\ et \geq 0,\ s \geq 0,\ r = 0 \right\}.$$

- **Radial geometric mean cone** - A cone in the power cone family defined by

$$\left\{ \begin{pmatrix} p \\ x \end{pmatrix} \in \mathbb{R}_+^k \times \mathbb{R}^1,\ \left(\prod_{j=1}^{k} p_j \right)^{\frac{1}{k}} \geq |x| \right\},\ \text{for } n = k+1 \geq 2.$$

- **Dual radial geometric mean cone** - A cone in the power cone family defined by

$$\left\{ \begin{pmatrix} p \\ x \end{pmatrix} \in \mathbb{R}_+^k \times \mathbb{R}^1,\ \left(\prod_{j=1}^{k} k p_j \right)^{\frac{1}{k}} \geq |x| \right\},\ \text{for } n = k+1 \geq 2.$$

and, the following parametric cones:

- **Radial power cone** - A cone in the power cone family defined by

$$\left\{ \begin{pmatrix} p \\ x \end{pmatrix} \in \mathbb{R}^k_+ \times \mathbb{R}^{n-k}, \left(\prod_{j=1}^{k} p_j^{\alpha_j} \right)^{\frac{1}{\sigma}} \geq \|x\|_2 \right\}, \text{ for } n \geq k \geq 1.$$

  where, $\sigma = \sum_{j=1}^{k} \alpha_j$ and $\alpha = \mathbb{R}^k_{++}$.

- **Dual radial power cone** - A cone in the power cone family defined by

$$\left\{ \begin{pmatrix} p \\ x \end{pmatrix} \in \mathbb{R}^k_+ \times \mathbb{R}^{n-k}, \left(\prod_{j=1}^{k} \left(\frac{\sigma p_j}{\alpha_j} \right)^{\alpha_j} \right)^{\frac{1}{\sigma}} \geq \|x\|_2 \right\}, \text{ for } n \geq k \geq 1.$$

  where, $\sigma = \sum_{j=1}^{k} \alpha_j$ and $\alpha = \mathbb{R}^k_{++}$.

14.4.2 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.

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.

14.4.3 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 14.3. 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 14.3.

- **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 sizes are given as follows.

Table 14.3: Cones available in the CBF format

Name	CBF keyword	Cone family	Cone size
Free domain	`F`	linear	$n \geq 1$
Positive orthant	`L+`	linear	$n \geq 1$
Negative orthant	`L-`	linear	$n \geq 1$
Fixpoint zero	`L=`	linear	$n \geq 1$
Quadratic cone	`Q`	second-order	$n \geq 1$
Rotated quadratic cone	`QR`	second-order	$n \geq 2$
Exponential cone	`EXP`	exponential	$n = 3$
Dual exponential cone	`EXP*`	exponential	$n = 3$
Radial geometric mean cone	`GMEANABS`	power	$n = k + 1 \geq 2$
Dual radial geometric mean cone	`GMEANABS*`	power	$n = k + 1 \geq 2$
Radial power cone (parametric)	`POW`	power	$n \geq k \geq 1$
Dual radial power cone (parametric)	`POW*`	power	$n \geq k \geq 1$

14.4.4 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.

POWCONES

Description: Define a lookup table for power cone domains.
`HEADER`: One line formatted as:

```
INT INT
```

This is the number of cones to be specified and the combined length of their dense parameter vectors.

`BODY`: **A list of chunks each specifying the dense parameter vector of a power cone.**
`CHUNKHEADER`: One line formatted as:

```
INT
```

This is the parameter vector length.

CHUNKBODY: A list of lines formatted as:

```
REAL
```

This is the parameter vector values. The number of lines should match the number stated in the chunk header.

The cone specified at index k (with 0-based indexing) is registered under the CBF name @k:POW.

POW*CONES

Description: Define a lookup table for dual power cone domains.
HEADER: One line formatted as:

```
INT INT
```

This is the number of cones to be specified and the combined length of their dense parameter vectors.

BODY: A list of chunks each specifying the dense parameter vector of a dual power cone.
CHUNKHEADER: One line formatted as:

```
INT
```

This is the parameter vector length.

CHUNKBODY: A list of lines formatted as:

```
REAL
```

This is the parameter vector values. The number of lines should match the number stated in the chunk header.

The cone specified at index k (with 0-based indexing) is registered under the CBF name @k:POW*.

OBJSENSE

Description: Define the objective sense.
HEADER: None
BODY: One line formatted as:

```
STR
```

having MIN indicates minimize, and MAX indicates maximize. Upper-case 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 14.3), 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 14.3), 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.

14.4.5 CBF Format Examples

Minimal Working Example

The conic optimization problem (14.10) , has three variables in a quadratic cone - first one is integer - and an affine expression in domain 0 (equality constraint).

$$\begin{array}{ll} \text{minimize} & 5.1\, x_0 \\ \text{subject to} & 6.2\, x_1 + 7.3\, x_2 - 8.4 \in \{0\} \\ & x \in \mathcal{Q}^3,\ x_0 \in \mathbb{Z}. \end{array} \tag{14.10}$$

Its formulation in the Conic Benchmark Format begins with the version of the CBF format used, to safeguard against later revisions.

```
VER
4
```

Next follows the problem structure, consisting of the objective sense, the number and domain of variables, the indices of integer variables, and the number and domain of scalar-valued affine expressions (i.e., the equality constraint).

```
OBJSENSE
MIN

VAR
3 1
Q 3

INT
1
0

CON
1 1
L= 1
```

Finally follows the problem data, consisting of the coefficients of the objective, the coefficients of the constraints, and the constant terms of the constraints. All data is specified on a sparse coordinate form.

```
OBJACOORD
1
0 5.1

ACOORD
2
0 1 6.2
0 2 7.3

BCOORD
1
0 -8.4
```

This concludes the example.

Mixing Linear, Second-order and Semidefinite Cones

The conic optimization problem (14.11), has a semidefinite cone, a quadratic cone over unordered subindices, and two equality constraints.

$$
\begin{array}{lll}
\text{minimize} & \left\langle \begin{bmatrix} 2 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 2 \end{bmatrix}, X_1 \right\rangle + x_1 & \\
\text{subject to} & \left\langle \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, X_1 \right\rangle + x_1 & = \quad 1.0\,, \\
& \left\langle \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, X_1 \right\rangle + x_0 + x_2 & = \quad 0.5\,, \\
& x_1 \geq \sqrt{x_0^2 + x_2^2}\,, & \\
& X_1 \succeq \mathbf{0}\,. &
\end{array}
\tag{14.11}
$$

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 4.
VER
4

# The sense of the objective is:
#     | Minimize.
OBJSENSE
MIN

# One PSD variable of this size:
#     | Three times three.
PSDVAR
1
3

# Three scalar variables in this one conic domain:
#     | Three are free.
VAR
3 1
F 3

# Five scalar constraints with affine expressions in two conic domains:
#     | Two are fixed to zero.
#     | Three are in conic quadratic domain.
CON
5 2
L= 2
Q 3

# Five coordinates in F^{obj}_j coefficients:
#     | F^{obj}[0][0,0] = 2.0
#     | F^{obj}[0][1,0] = 1.0
#     | and more...
OBJFCOORD
5
```

(continues on next page)

(continued from previous page)

```
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

# 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.

$$
\begin{array}{lllll}
\text{minimize} & \left\langle \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, X_1 \right\rangle + x_1 + x_2 + 1 & & & \\
\text{subject to} & \left\langle \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, X_1 \right\rangle - x_1 - x_2 & \geq & 0.0\,, \\
 & x_1 \begin{bmatrix} 0 & 1 \\ 1 & 3 \end{bmatrix} + x_2 \begin{bmatrix} 3 & 1 \\ 1 & 0 \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \succeq & \mathbf{0}\,, \\
 & X_1 \succeq \mathbf{0}\,. & &
\end{array}
\tag{14.12}
$$

Its formulation in the CBF format is written in what follows

```
# File written using this version of the Conic Benchmark Format:
#     | Version 4.
VER
4

# The sense of the objective is:
#     | Minimize.
OBJSENSE
MIN

# One PSD variable of this size:
#     | Two times two.
PSDVAR
1
2

# Two scalar variables in this one conic domain:
#     | Two are free.
VAR
2 1
F 2

# One PSD constraint of this size:
#     | Two times two.
PSDCON
1
2

# One scalar constraint with an affine expression in this one conic domain:
#     | One is greater than or equal to zero.
CON
1 1
L+ 1

# Two coordinates in F^{obj}_j coefficients:
#     | F^{obj}[0][0,0] = 1.0
#     | F^{obj}[0][1,1] = 1.0
OBJFCOORD
2
0 0 0 1.0
0 1 1 1.0

# Two coordinates in a^{obj}_j coefficients:
```

(continues on next page)

(continued from previous page)

```
#     | 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
```

The exponential cone

The conic optimization problem (14.13), has one equality constraint, one quadratic cone constraint and an exponential cone constraint.

$$\begin{array}{ll}\text{minimize} & x_0 - x_3\\ \text{subject to} & x_0 + 2\,x_1 - x_2 \in \{0\}\\ & (5.0, x_0, x_1) \in \mathcal{Q}^3\\ & (x_2, 1.0, x_3) \in EXP.\end{array} \tag{14.13}$$

The nonlinear conic constraints enforce $\sqrt{x_0^2 + x_1^2} \leq 0.5$ and $x_3 \leq \log(x_2)$.

```
# File written using this version of the Conic Benchmark Format:
#       | Version 3.
VER
3

# The sense of the objective is:
#       | Minimize.
OBJSENSE
MIN

# Four scalar variables in this one conic domain:
#       | Four are free.
VAR
4 1
F 4

# Seven scalar constraints with affine expressions in three conic domains:
#       | One is fixed to zero.
#       | Three are in conic quadratic domain.
#       | Three are in exponential cone domain.
CON
7 3
L= 1
Q 3
EXP 3

# Two coordinates in a^{obj}_j coefficients:
#       | a^{obj}[0] = 1.0
#       | a^{obj}[3] = -1.0
OBJACOORD
2
0 1.0
3 -1.0

# Seven coordinates in a_ij coefficients:
#       | a[0,0] = 1.0
#       | a[0,1] = 2.0
#       | and more...
ACOORD
7
0 0 1.0
0 1 2.0
0 2 -1.0
2 0 1.0
3 1 1.0
4 2 1.0
6 3 1.0
```

(continues on next page)

(continued from previous page)

```
# Two coordinates in b_i coefficients:
#       | b[1] = 5.0
#       | b[5] = 1.0
BCOORD
2
1 5.0
5 1.0
```

Parametric cones

The problem (14.14), has three variables in a power cone with parameter $\alpha_1 = (1, 1)$ and two power cone constraints each with parameter $\alpha_0 = (8, 1)$.

$$\begin{array}{ll} \text{minimize} & x_3 \\ \text{subject to} & (1.0, x_1, x_1 + x_2) \in POW_{\alpha_0} \\ & (1.0, x_2, x_1 + x_2) \in POW_{\alpha_0} \\ & x \in POW_{\alpha_1}. \end{array} \tag{14.14}$$

The nonlinear conic constraints enforce $x_3 \leq x_1 x_2$ and $x_1 + x_2 \leq \min(x_1^{\frac{1}{9}}, x_2^{\frac{1}{9}})$.

```
# File written using this version of the Conic Benchmark Format:
#       | Version 3.
VER
3

# Two power cone domains defined in a total of four parameters:
#       | @0:POW (specification 0) has two parameters:
#       | alpha[0] = 8.0.
#       | alpha[1] = 1.0.
#       | @1:POW (specification 1) has two parameters:
#       | alpha[0] = 1.0.
#       | alpha[1] = 1.0.
POWCONES
2 4
2
8.0
1.0
2
1.0
1.0

# The sense of the objective is:
#       | Maximize.
OBJSENSE
MAX

# Three scalar variable in this one conic domain:
#       | Three are in power cone domain (specification 1).
VAR
3 1
@1:POW 3

# Six scalar constraints with affine expressions in two conic domains:
#       | Three are in power cone domain (specification 0).
#       | Three are in power cone domain (specification 0).
```

(continues on next page)

(continued from previous page)

```
CON
6 2
@0:POW 3
@0:POW 3

# One coordinate in a^{obj}_j coefficients:
#      | a^{obj}[2] = 1.0
OBJACOORD
1
2 1.0

# Six coordinates in a_ij coefficients:
#      | a[1,0] = 1.0
#      | a[2,0] = 1.0
#      | and more...
ACOORD
6
1 0 1.0
2 0 1.0
2 1 1.0
4 1 1.0
5 0 1.0
5 1 1.0

# Two coordinates in b_i coefficients:
#      | b[0] = 1.0
#      | b[3] = 1.0
BCOORD
2
0 1.0
3 1.0
```

14.5 The PTF Format

The PTF format is a human-readable, natural text format that supports all linear, conic and mixed-integer features.

14.5.1 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: LINEAR_TERM | MATRIX_TERM
```

A linear term

```
LINEAR_TERM: FLOAT? NAME
```

A matrix term

```
MATRIX_TERM: "<" ( [+-] 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 matrices. The actual matrices 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
```

14.5.2 `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.

14.5.3 `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 >
```

14.5.4 `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
  # Ranged constraint
  'c1' [0;10] + x1 + x2 + x3
  # Fixed constraint, expression equals to 0
  [0] + x1 + x2 + x3
  # Nonnegative constraint
  [0;+inf] + 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:

- **Major (primal) cones:**
  - `QUAD(N)` or `SOC(N)`: Second order cone of dimension `N`.
  - `RQUAD(N)` or `RSOC(N)`: Rotated second order cone of dimension `N`.
  - `PEXP`: Primal exponential cone of dimension 3.
  - `PPOW(N,P)`: Primal power cone of dimension `N` with parameter `P` (float between 0 and 1).
  - `PPOW(N;ALPHA)`: Primal power cone of dimension `N` with exponent sequence `ALPHA` (comma-separated list of floats).
  - `PGEOMEAN(N)`: Primal geometric mean cone of domension `N`.
  - `SVECPSD(N)`: Vectorized symmetric positive semidefinite cone of dimension `N` (`N` must be of the form `D*(D+1)/2`).
- **Dual cones:**
  - `DEXP`: Dual exponential cone of dimension 3.
  - `DPOW(N,P)`: Dual power cone of dimension `N` with parameter `P` (float between 0 and 1).
  - `DPOW(N;ALPHA)`: Dual power cone of dimension `N` with exponent sequence `ALPHA` (comma-separated list of floats).
  - `DGEOMEAN(N)`: Dual geometric mean cone of domension `N`.
- **Linear cones:**
  - `FREE(N)` The free (unbounded) cone of dimension `N`.
  - `POSITIVE(N)` The non-negative cone of dimension `N`.

- NEGATIVE(N) The non-positive cone of dimension N.
- ZERO(N) The zero-cone of dimension N.

See Sec. 13.8 for definitions of the parameters.

```
"Constraints"
  NAME? "[" DOMAIN "]" EXPR_LIST
```

For example:

```
Constraints
   'K1' [PPOW(5;3,1)]
      + x1 + x2
      + x2 + x3
      + 1.0
      + x1
      + x3
   'K2' [RQUAD(3)]
      + x1 + x2
      + x2 + x3
      + x3 + x1
```

14.5.5 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 "[" "PSD" (INT) "]"
```

For example, a linear variable

```
Variables
   # Nonnegative variable
   x1 [0;inf]
   # Ranged variable
   x2 [0;1]
   # Fixed variable
   x3 [5.0]
   # 5-dimensional symmetric matrix variable
   X [PSD(5)]
```

14.5.6 `Integer` section

This section contains a list of variables that are integral. For example:

```
Integer
   x1 x2 x3
```

14.5.7 `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)
```

14.5.8 `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_OR_NONE
   "Variables"
      # Linear variable status: level, slx, sux
      NAME "[" STATUS "]" FLOAT FLOAT_OR_NONE FLOAT_OR_NONE
   "Constraints"
      # Linear variable status: level, slx, sux
      NAME "[" STATUS "]" FLOAT FLOAT_OR_NONE FLOAT_OR_NONE
      # Conic constraint status: level, doty
      NAME
         "[" STATUS "]" FLOAT FLOAT_OR_NONE
```

Nonexistent values (for example, dual values for an integer solution) are replaced with a single dot (`.`):

```
FLOAT_OR_NONE = 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

14.5.9 Examples

Linear example `lo1.ptf`

```
Task ''
    # Written by MOSEK v10.0.13
    # problemtype: Linear Problem
    # number of linear variables: 4
    # number of linear constraints: 3
    # number of old-style A nonzeros: 9
Objective obj
    Maximize + 3 x1 + x2 + 5 x3 + x4
Constraints
    c1 [3e+1] + 3 x1 + x2 + 2 x3
    c2 [1.5e+1;+inf] + 2 x1 + x2 + 3 x3 + x4
    c3 [-inf;2.5e+1] + 2 x2 + 3 x4
Variables
    x1 [0;+inf]
    x2 [0;1e+1]
    x3 [0;+inf]
    x4 [0;+inf]
```

Conic quadratic example cqo1.ptf

```
Task ''
    # Written by MOSEK v10.0.17
    # problemtype: Conic Problem
    # number of linear variables: 6
    # number of linear constraints: 1
    # number of  old-style cones: 0
    # number of positive semidefinite variables: 0
    # number of positive semidefinite matrixes: 0
    # number of affine conic constraints: 2
    # number of disjunctive constraints: 0
    # number scalar affine expressions/nonzeros : 6/6
    # number of old-style A nonzeros: 3
Objective obj
    Minimize + x4 + x5 + x6
Constraints
    c1 [1] + x1 + x2 + 2 x3
    k1 [QUAD(3)]
        @ac1: + x4
        @ac2: + x1
        @ac3: + x2
    k2 [RQUAD(3)]
        @ac4: + x5
        @ac5: + x6
        @ac6: + x3
Variables
    x4
    x1 [0;+inf]
    x2 [0;+inf]
    x5
    x6
    x3 [0;+inf]
```

Power cone example cqo1.ptf

```
Task ''
Objective ''
    Maximize - x0 + x3 + x4
Constraints
    c0 [2] + x0 + x1 + 5e-1 x2
    C1 [PPOW(3,2e-1)]
        + x0
        + x1
        + x3
    C2 [PPOW(3;4.0,6.0)]
        + x2
        + x5
        + x4
Variables
    x0
    x1
    x2
    x3
    x4
    x5 [1.0]
```

Disjunctive example djc1.ptf

```
Task djc1
Objective ''
    Minimize + 2 'x[0]' + 'x[1]' + 3 'x[2]' + 'x[3]'
Constraints
    @c0 [-10;+inf] + 'x[0]' + 'x[1]' + 'x[2]' + 'x[3]'
    @D0 [OR]
        [AND]
            [NEGATIVE(1)]
                 + 'x[0]' - 2 'x[1]' + 1
            [ZERO(2)]
                 + 'x[2]'
                 + 'x[3]'
        [AND]
            [NEGATIVE(1)]
                 + 'x[2]' - 3 'x[3]' + 2
            [ZERO(2)]
                 + 'x[0]'
                 + 'x[1]'
    @D1 [OR]
        [ZERO(1)]
             + 'x[0]' - 2.5
        [ZERO(1)]
             + 'x[1]' - 2.5
        [ZERO(1)]
             + 'x[2]' - 2.5
        [ZERO(1)]
             + 'x[3]' - 2.5
Variables
    'x[0]'
    'x[1]'
    'x[2]'
    'x[3]'
```

Semidefinite example sdo1.ptf

```
Task ''
    # Written by MOSEK v10.0.17
    # problemtype: Conic Problem
    # number of linear variables: 3
    # number of linear constraints: 0
    # number of  old-style cones: 0
    # number of positive semidefinite variables: 1
    # number of positive semidefinite matrixes: 3
    # number of affine conic constraints: 2
    # number of disjunctive constraints: 0
    # number scalar affine expressions/nonzeros : 5/6
    # number of old-style A nonzeros: 0
Objective ''
    Minimize + @x0 + <M0;@X0>
Constraints
    @C0 [ZERO(2)]
        @ac0: + @x0 + < + M1;@X0> - 1
        @ac1: + @x1 + @x2 + < + M2;@X0> - 0.5
    @C1 [QUAD(3)]
```

(continues on next page)

(continued from previous page)

```
        @ac2: + @x0
        @ac3: + @x1
        @ac4: + @x2
Variables
    @x0
    @x1
    @x2
    @X0 [PSD(3)]
SymmetricMatrixes
    M0 SYMMAT(3) (0,0,2) (1,0,1) (1,1,2) (2,1,1) (2,2,2)
    M1 SYMMAT(3) (0,0,1) (1,1,1) (2,2,1)
    M2 SYMMAT(3) (0,0,1) (1,0,1) (1,1,1) (2,0,1) (2,1,1) (2,2,1)
```

14.6 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.

14.7 The JSON Format

MOSEK provides the possibility to read/write problems and solutions in JSON format. The official JSON website http://www.json.org provides plenty of information along with the format definition. JSON is an industry standard for data exchange and JSON files can be easily written and read in most programming languages using dedicated libraries.

MOSEK uses two JSON-based formats:

- **JTASK**, for storing problem instances together with solutions and parameters. The JTASK format contains the same information as a native **MOSEK** task *task format*, that is a very close representation of the internal data storage in the task object.

  You can write a JTASK file specifying the extension `.jtask`. When the parameter *MSK_IPAR_WRITE_JSON_INDENTATION* is set the JTASK file will be indented to slightly improve readability.

- **JSOL**, for storing solutions and information items.

14.7.1 JTASK Specification

The JTASK is a dictionary containing the following sections. All sections are optional and can be omitted if irrelevant for the problem.

- `$schema`: JSON schema.
- `Task/name`: The name of the task (string).
- `Task/INFO`: Information about problem data dimensions and similar. These are treated as hints when reading the file.
  - `numvar`: number of variables (int32).
  - `numcon`: number of constraints (int32).
  - `numcone`: number of cones (int32, deprecated).
  - `numbarvar`: number of symmetric matrix variables (int32).
  - `numanz`: number of nonzeros in A (int64).
  - `numsymmat`: number of matrices in the symmetric matrix storage E (int64).
  - `numafe`: number of affine expressions in AFE storage (int64).
  - `numfnz`: number of nonzeros in F (int64).
  - `numacc`: number of affine conic constraints (ACCs) (int64).
  - `numdjc`: number of disjunctive constraints (DJCs) (int64).
  - `numdom`: number of domains (int64).
  - `mosekver`: MOSEK version (list(int32)).
- `Task/data`: Numerical and structural data of the problem.
  - `var`: Information about variables. All fields present must have the same length as `bk`. All or none of `bk`, `bl`, and `bu` must appear.
    - `name`: Variable names (list(string)).
    - `bk`: Bound keys (list(string)).
    - `bl`: Lower bounds (list(double)).
    - `bu`: Upper bounds (list(double)).
    - `type`: Variable types (list(string)).
  - `con`: Information about linear constraints. All fields present must have the same length as `bk`. All or none of `bk`, `bl`, and `bu` must appear.
    - `name`: Constraint names (list(string)).
    - `bk`: Bound keys (list(string)).
    - `bl`: Lower bounds (list(double)).
    - `bu`: Upper bounds (list(double)).
  - `barvar`: Information about symmetric matrix variables. All fields present must have the same length as `dim`.
    - `name`: Barvar names (list(string)).
    - `dim`: Dimensions (list(int32)).
  - `objective`: Information about the objective.
    - `name`: Objective name (string).
    - `sense`: Objective sense (string).
    - `c`: The linear part c of the objective as a sparse vector. Both arrays must have the same length.
      - `subj`: indices of nonzeros (list(int32)).
      - `val`: values of nonzeros (list(double)).
    - `cfix`: Constant term in the objective (double).

    * `Q`: The quadratic part Q^o of the objective as a sparse matrix, only lower-triangular part included. All arrays must have the same length.
        · `subi`: row indices of nonzeros (list(int32)).
        · `subj`: column indices of nonzeros (list(int32)).
        · `val`: values of nonzeros (list(double)).
    * `barc`: The semidefinite part $\overline{C}$ of the objective (list). Each element of the list is a list describing one entry $\overline{C}_j$ using three fields:
        · index j (int32).
        · weights of the matrices from the storage E forming $\overline{C}_j$ (list(double)).
        · indices of the matrices from the storage E forming $\overline{C}_j$ (list(int64)).
- `A`: The linear constraint matrix A as a sparse matrix. All arrays must have the same length.
    * `subi`: row indices of nonzeros (list(int32)).
    * `subj`: column indices of nonzeros (list(int32)).
    * `val`: values of nonzeros (list(double)).
- `bara`: The semidefinite part $\overline{A}$ of the constraints (list). Each element of the list is a list describing one entry $\overline{A}_{ij}$ using four fields:
    * index i (int32).
    * index j (int32).
    * weights of the matrices from the storage E forming $\overline{A}_{ij}$ (list(double)).
    * indices of the matrices from the storage E forming $\overline{A}_{ij}$ (list(int64)).
- `AFE`: The affine expression storage.
    * `numafe`: number of rows in the storage (int64).
    * `F`: The matrix F as a sparse matrix. All arrays must have the same length.
        · `subi`: row indices of nonzeros (list(int64)).
        · `subj`: column indices of nonzeros (list(int32)).
        · `val`: values of nonzeros (list(double)).
    * `g`: The vector g of constant terms as a sparse vector. Both arrays must have the same length.
        · `subi`: indices of nonzeros (list(int64)).
        · `val`: values of nonzeros (list(double)).
    * `barf`: The semidefinite part $\overline{F}$ of the expressions in AFE storage (list). Each element of the list is a list describing one entry $\overline{F}_{ij}$ using four fields:
        · index i (int64).
        · index j (int32).
        · weights of the matrices from the storage E forming $\overline{F}_{ij}$ (list(double)).
        · indices of the matrices from the storage E forming $\overline{F}_{ij}$ (list(int64)).
- `domains`: Information about domains. All fields present must have the same length as `type`.
    * `name`: Domain names (list(string)).
    * `type`: Description of the type of each domain (list). Each element of the list is a list describing one domain using at least one field:
        · domain type (string).
        · (except `pexp`, `dexp`) dimension (int64).
        · (only `ppow`, `dpow`) weights (list(double)).
- `ACC`: Information about affine conic constraints (ACC). All fields present must have the same length as `domain`.
    * `name`: ACC names (list(string)).
    * `domain`: Domains (list(int64)).
    * `afeidx`: AFE indices, grouped by ACC (list(list(int64))).
    * `b`: constant vectors b, grouped by ACC (list(list(double))).

- DJC: Information about disjunctive constraints (DJC). All fields present must have the same length as termsize.
  - name: DJC names (list(string)).
  - termsize: Term sizes, grouped by DJC (list(list(int64))).
  - domain: Domains, grouped by DJC (list(list(int64))).
  - afeidx: AFE indices, grouped by DJC (list(list(int64))).
  - b: constant vectors b, grouped by DJC (list(list(double))).
- MatrixStore: The symmetric matrix storage E (list). Each element of the list is a list describing one entry E using four fields in sparse matrix format, lower-triangular part only:
  - dimension (int32).
  - row indices of nonzeros (list(int32)).
  - column indices of nonzeros (list(int32)).
  - values of nonzeros (list(double)).
- Q: The quadratic part Q^c of the constraints (list). Each element of the list is a list describing one entry Q^c_i using four fields in sparse matrix format, lower-triangular part only:
  - the row index i (int32).
  - row indices of nonzeros (list(int32)).
  - column indices of nonzeros (list(int32)).
  - values of nonzeros (list(double)).
- qcone (deprecated). The description of cones. All fields present must have the same length as type.
  - name: Cone names (list(string)).
  - type: Cone types (list(string)).
  - par: Additional cone parameters (list(double)).
  - members: Members, grouped by cone (list(list(int32))).

- Task/solutions: Solutions. This section can contain up to three subsections called:
  - interior
  - basic
  - integer

  corresponding to the three solution types in MOSEK. Each of these sections has the same structure:
  - prosta: problem status (string).
  - solsta: solution status (string).
  - xx, xc, y, slc, suc, slx, sux, snx: one for each component of the solution of the same name (list(double)).
  - skx, skc, skn: status keys (list(string)).
  - doty: the dual $\dot{y}$ solution, grouped by ACC (list(list(double))).
  - barx, bars: the primal/dual semidefinite solution, grouped by matrix variable (list(list(double))).

- Task/parameters: Parameters.
  - iparam: Integer parameters (dictionary). A dictionary with entries of the form name:value, where name is a shortened parameter name (without leading MSK_IPAR_) and value is either an integer or string if the parameter takes values from an enum.
  - dparam: Double parameters (dictionary). A dictionary with entries of the form name:value, where name is a shortened parameter name (without leading MSK_DPAR_) and value is a double.
  - sparam: String parameters (dictionary). A dictionary with entries of the form name:value, where name is a shortened parameter name (without leading MSK_SPAR_) and value is a string. Note that this section is allowed but MOSEK ignores it both when writing and reading JTASK files.

14.7.2 JSOL Specification

The JSOL is a dictionary containing the following sections. All sections are optional and can be omitted if irrelevant for the problem.

- `$schema`: JSON schema.
- `Task/name`: The name of the task (string).
- `Task/solutions`: Solutions. This section can contain up to three subsections called:
  - `interior`
  - `basic`
  - `integer`

  corresponding to the three solution types in MOSEK. Each of these section has the same structure:
  - `prosta`: problem status (string).
  - `solsta`: solution status (string).
  - `xx`, `xc`, `y`, `slc`, `suc`, `slx`, `sux`, `snx`: one for each component of the solution of the same name (list(double)).
  - `skx`, `skc`, `skn`: status keys (list(string)).
  - `doty`: the dual $\dot{y}$ solution, grouped by ACC (list(list(double))).
  - `barx`, `bars`: the primal/dual semidefinite solution, grouped by matrix variable (list(list(double))).
- `Task/information`: Information items from the optimizer.
  - `int32`: int32 information items (dictionary). A dictionary with entries of the form `name: value`.
  - `int64`: int64 information items (dictionary). A dictionary with entries of the form `name: value`.
  - `double`: double information items (dictionary). A dictionary with entries of the form `name: value`.

14.7.3 A `jtask` example

Listing 14.5: A formatted `jtask` file for a simple portfolio optimization problem.

```
{
   "$schema":"http://mosek.com/json/schema#",
   "Task/name":"Markowitz portfolio with market impact",
   "Task/INFO":{"numvar":7,"numcon":1,"numcone":0,"numbarvar":0,"numanz":6,"numsymmat
↪":0,"numafe":13,"numfnz":12,"numacc":4,"numdjc":0,"numdom":3,"mosekver":[10,0,0,3]},
   "Task/data":{
      "var":{
         "name":["1.0","x[0]","x[1]","x[2]","t[0]","t[1]","t[2]"],
         "bk":["fx","lo","lo","lo","fr","fr","fr"],
         "bl":[1,0.0,0.0,0.0,-1e+30,-1e+30,-1e+30],
         "bu":[1,1e+30,1e+30,1e+30,1e+30,1e+30,1e+30],
         "type":["cont","cont","cont","cont","cont","cont","cont"]
      },
      "con":{
         "name":["budget[]"],
         "bk":["fx"],
         "bl":[1],
```

(continues on next page)

(continued from previous page)

```
            "bu":[1]
        },
        "objective":{
            "sense":"max",
            "name":"obj",
            "c":{
                "subj":[1,2,3],
                "val":[0.1073,0.0737,0.0627]
            },
            "cfix":0.0
        },
        "A":{
            "subi":[0,0,0,0,0,0],
            "subj":[1,2,3,4,5,6],
            "val":[1,1,1,0.01,0.01,0.01]
        },
        "AFE":{
            "numafe":13,
            "F":{
                "subi":[1,1,1,2,2,3,4,6,7,9,10,12],
                "subj":[1,2,3,2,3,3,4,1,5,2,6,3],
                "val":[0.166673333200005,0.0232190712557243,0.0012599496030238,0.
↪102863378954911,-0.00222873156550421,0.0338148677744977,1,1,1,1,1,1]
            },
            "g":{
                "subi":[0,5,8,11],
                "val":[0.035,1,1,1]
            }
        },
        "domains":{
            "type":[["r",0],
                    ["quad",4],
                    ["ppow",3,[0.6666666666666666,0.33333333333333337]]]
        },
        "ACC":{
            "name":["risk[]","tz[0]","tz[1]","tz[2]"],
            "domain":[1,2,2,2],
            "afeidx":[[0,1,2,3],
                      [4,5,6],
                      [7,8,9],
                      [10,11,12]]
        }
    },
    "Task/solutions":{
        "interior":{
            "prosta":"unknown",
            "solsta":"unknown",
            "skx":["fix","supbas","supbas","supbas","supbas","supbas","supbas"],
            "skc":["fix"],
            "xx":[1,0.10331580274282556,0.11673185566457132,0.7724326587076371,0.
↪033208600335718846,0.03988270849469869,0.6788769587942524],
            "xc":[1],
            "slx":[0.0,-5.585840467641202e-10,-8.945844685006369e-10,-7.815248786428623e-
↪11,0.0,0.0,0.0],
            "sux":[0.0,0.0,0.0,0.0,0.0,0.0,0.0],
            "snx":[0.0,0.0,0.0,0.0,0.0,0.0,0.0],
```

(continues on next page)

(continued from previous page)


```
         "slc":[0.0],
         "suc":[-0.046725814048521205],
         "y":[0.046725814048521205],
         "doty":[[-0.6062603164682975,0.3620818321879349,0.17817754087278295,0.
↪4524390346223723],
                 [-4.6725842015519993e-4,-7.708781121860897e-6,2.24800624747081e-4],
                 [-4.6725842015519993e-4,-9.268264309496919e-6,2.390390600079771e-4],
                 [-4.6725842015519993e-4,-1.5854982159992136e-4,6.159249331148646e-4]]
      }
   },
   "Task/parameters":{
      "iparam":{
         "LICENSE_DEBUG":"ON",
         "MIO_SEED":422
      },
      "dparam":{
         "MIO_MAX_TIME":100
      },
      "sparam":{
      }
   }
}
```


14.8 The Solution File Format

MOSEK can output solutions to a text file:

- *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 is solved with the mixed-integer optimizer.

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>

AFFINE CONIC CONSTRAINTS
INDEX   NAME     I          ACTIVITY    DUAL
?       <name>   <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>]
```

(continues on next page)

(continued from previous page)

```
SYMMETRIC MATRIX VARIABLES
INDEX  NAME     I          J            PRIMAL       DUAL
?      <name>   <a value>  <a value>    <a value>    <a value>
```

The fields `?`, `??` and `<>` will be filled with problem and solution specific information as described below. The solution contains sections corresponding to parts of the input. Empty sections may be omitted and fields in `[]` are optional, depending on what type of problem is solved.

- `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`
  - `INDEX`: A sequential index assigned to the constraint by **MOSEK**
  - `NAME`: The name of the constraint assigned by the user or autogenerated.
  - `AT`: The status key `bkc` of the constraint as in Table 14.4.
  - `ACTIVITY`: the activity `xc` of the constraint expression.
  - `LOWER LIMIT`: the lower bound `blc` of the constraint.
  - `UPPER LIMIT`: the upper bound `buc` of the constraint.
  - `DUAL LOWER`: the dual multiplier `slc` corresponding to the lower limit on the constraint.
  - `DUAL UPPER`: the dual multiplier `suc` corresponding to the upper limit on the constraint.
- `AFFINE CONIC CONSTRAINTS`
  - `INDEX`: A sequential index assigned to the affine expressions by **MOSEK**
  - `NAME`: The name of the affine conic constraint assigned by the user or autogenerated.
  - `I`: The sequential index of the affine expression in the affine conic constraint.
  - `ACTIVITY`: the activity of the `I`-th affine expression in the affine conic constraint.
  - `DUAL`: the dual multiplier `doty` for the `I`-th entry in the affine conic constraint.
- `VARIABLES`
  - `INDEX`: A sequential index assigned to the variable by **MOSEK**
  - `NAME`: The name of the variable assigned by the user or autogenerated.
  - `AT`: The status key `bkx` of the variable as in Table 14.4.
  - `ACTIVITY`: the value `xx` of the variable.
  - `LOWER LIMIT`: the lower bound `blx` of the variable.
  - `UPPER LIMIT`: the upper bound `bux` of the variable.
  - `DUAL LOWER`: the dual multiplier `slx` corresponding to the lower limit on the variable.
  - `DUAL UPPER`: the dual multiplier `sux` corresponding to the upper limit on the variable.
  - `CONIC DUAL`: the dual multiplier `skx` corresponding to a conic variable (deprecated).
- `SYMMETRIC MATRIX VARIABLES`
  - `INDEX`: A sequential index assigned to each symmetric matrix entry by **MOSEK**
  - `NAME`: The name of the symmetric matrix variable assigned by the user or autogenerated.
  - `I`: The row index in the symmetric matrix variable.
  - `J`: The column index in the symmetric matrix variable.
  - `PRIMAL`: the value of `barx` for the (`I`, `J`)-th entry in the symmetric matrix variable.
  - `DUAL`: the dual multiplier `bars` for the (`I`, `J`)-th entry in the symmetric matrix variable.

Table 14.4: 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.

Example.

Below is an example of a solution file.

Listing 14.6: An example of .sol file.

```
NAME                    :
PROBLEM STATUS          : PRIMAL_AND_DUAL_FEASIBLE
SOLUTION STATUS         : OPTIMAL
OBJECTIVE NAME          : OBJ
PRIMAL OBJECTIVE        : 0.70571049347734
DUAL OBJECTIVE          : 0.70571048919757

CONSTRAINTS
INDEX      NAME           AT ACTIVITY                 LOWER LIMIT        UPPER LIMIT ␣
↪       DUAL LOWER              DUAL UPPER

AFFINE CONIC CONSTRAINTS
INDEX      NAME           I          ACTIVITY                  DUAL
0          A1             0          1.0000000009656           0.54475821296644
1          A1             1          0.50000000152223          0.32190455246225
2          A2             0          0.25439922724695          0.4552417870329
3          A2             1          0.17988741850378          -0.32190455246178
4          A2             2          0.17988741850378          -0.32190455246178

VARIABLES
INDEX      NAME           AT ACTIVITY                 LOWER LIMIT        UPPER LIMIT ␣
↪       DUAL LOWER              DUAL UPPER
0          X1             SB 0.25439922724695         NONE               NONE        ␣
↪       0                       0
1          X2             SB 0.17988741850378         NONE               NONE        ␣
↪       0                       0
2          X3             SB 0.17988741850378         NONE               NONE        ␣
↪       0                       0

SYMMETRIC MATRIX VARIABLES
INDEX      NAME           I          J         PRIMAL                    DUAL
0          BARX1          0          0         0.21725733689874          1.1333372337141
1          BARX1          1          0         -0.25997257078534         0.
↪67809544651396
2          BARX1          2          0         0.21725733648507          -0.
↪3219045527104
3          BARX1          1          1         0.31108610088839          1.1333372332693
4          BARX1          2          1         -0.25997257078534         0.
↪67809544651435
5          BARX1          2          2         0.21725733689874          1.1333372337145
```

(continues on next page)

(continued from previous page)

```
6          BARX2          0          0          4.8362272828127e-10      0.
↪54475821339698
7          BARX2          1          0          0                        0
8          BARX2          1          1          4.8362272828127e-10      0.
↪54475821339698
```

Chapter 15

List of examples

List of examples shipped in the distribution of API for MATLAB:

Table 15.1: List of distributed examples

File	Description
acc1.m	A simple problem with one affine conic constraint (ACC)
affco1.m	A simple problem using affine conic constraints
affco2.m	A simple problem using affine conic constraints
ceo1.m	A simple conic exponential problem
cqo1.m	A simple conic quadratic problem
diet.m	Solving Stigler's Nutrition model diet from the GAMS model library
djc1.m	A simple problem with disjunctive constraints (DJC)
facility_locatio m	Demonstrates a small one-facility location problem (CQO)
gp1.m	A simple geometric program (GP) in conic form
helloworld.m	A Hello World example
lo1.m	A simple linear problem using msklpopt
lo1_simplex.m	A simple linear problem solved with the simplex solver
lo2.m	A simple linear problem using mosekopt
lo2_simplex.m	A simple linear problem solved with the simplex solver
lo_reoptimize. m	Simplex warm-start through reoptimization
lo_warmstart.m	Simplex warm-start with initial data
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
opt_server_sync. m	Uses **MOSEK** OptServer to solve an optimization problem synchronously
parameters.m	Shows how to set optimizer parameters and read information items
pinfeas.m	Shows how to obtain and analyze a primal infeasibility certificate
portfolio_1_basi m	Portfolio optimization - basic Markowitz model
portfolio_2_fron m	Portfolio optimization - efficient frontier
portfolio_3_impa m	Portfolio optimization - market impact costs
portfolio_4_tran m	Portfolio optimization - transaction costs
portfolio_5_card m	Portfolio optimization - cardinality constraints
portfolio_6_fact m	Portfolio optimization - factor model

continues on next page

Table 15.1 – continued from previous page

File	Description
pow1.m	A simple power cone problem
reoptimization.m	Demonstrate how to modify and re-optimize a linear problem
response.m	Demonstrates proper response handling

Additional examples can be found on the **MOSEK** website and in other **MOSEK** publications.

Chapter 16

Interface changes

The section shows interface-specific changes to the **MOSEK** API for MATLAB in version 11.1 compared to version 10. See the release notes for general changes and new features of the **MOSEK** Optimization Suite.

16.1 Important changes compared to version 10

- **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.
- **New API.** The API for MATLAB is a new API still under development. In the long run it will replace the previous toolbox and we encourage all users to move away from the old toolbox to this API.
- **Features.** In the present introductory version the API for MATLAB supports linear, conic (non-SDP), mixed-integer and disjunctive problems. Semidefinite problems will be supported in a future version.

16.2 Changes compared to version 10

16.2.1 Parameters compared to version 10

Added

- *MSK_DPAR_FOLDING_TOL_EQ*
- *MSK_DPAR_MIO_CLIQUE_TABLE_SIZE_FACTOR*
- *MSK_DPAR_SIM_PRECISION_SCALING_EXTENDED*
- *MSK_DPAR_SIM_PRECISION_SCALING_NORMAL*
- *MSK_IPAR_FOLDING_USE*
- *MSK_IPAR_GETDUAL_CONVERT_LMIS*
- *MSK_IPAR_HEARTBEAT_SIM_FREQ_TICKS*
- *MSK_IPAR_LOG_SIM_FREQ_GIGA_TICKS*
- *MSK_IPAR_MIO_CONFLICT_ANALYSIS_LEVEL*
- *MSK_IPAR_MIO_CROSSOVER_MAX_NODES*
- *MSK_IPAR_MIO_INDEPENDENT_BLOCK_LEVEL*

- *MSK_IPAR_MIO_OPT_FACE_MAX_NODES*
- *MSK_IPAR_MIO_RENS_MAX_NODES*
- *MSK_IPAR_PTF_WRITE_SINGLE_PSD_TERMS*
- *MSK_IPAR_READ_ASYNC*
- *MSK_IPAR_SIM_PRECISION*
- *MSK_IPAR_SIM_PRECISION_BOOST*
- *MSK_IPAR_WRITE_ASYNC*

Removed

- dparam.check_convexity_rel_tol
- dparam.presolve_tol_aij
- iparam.infeas_prefer_primal
- iparam.intpnt_max_num_refinement_steps
- iparam.intpnt_purify
- iparam.log_response
- iparam.log_sim_minor
- iparam.mio_root_repeat_presolve_level
- iparam.presolve_level
- iparam.sensitivity_optimizer
- iparam.sim_stability_priority
- iparam.sol_filter_keep_ranged
- iparam.solution_callback
- iparam.write_data_param
- iparam.write_generic_names_io
- iparam.write_task_inc_sol
- iparam.write_xml_mode
- sparam.write_lp_gen_var_name

16.2.2 Constants compared to version 10

Added

- *"MSK_CALLBACK_BEGIN_FOLDING"*
- *"MSK_CALLBACK_BEGIN_FOLDING_BI"*
- *"MSK_CALLBACK_BEGIN_FOLDING_BI_DUAL"*
- *"MSK_CALLBACK_BEGIN_FOLDING_BI_INITIALIZE"*
- *"MSK_CALLBACK_BEGIN_FOLDING_BI_OPTIMIZER"*

- *"MSK_CALLBACK_BEGIN_FOLDING_BI_PRIMAL"*
- *"MSK_CALLBACK_BEGIN_INITIALIZE_BI"*
- *"MSK_CALLBACK_BEGIN_OPTIMIZE_BI"*
- *"MSK_CALLBACK_DECOMP_MIO"*
- *"MSK_CALLBACK_END_FOLDING"*
- *"MSK_CALLBACK_END_FOLDING_BI"*
- *"MSK_CALLBACK_END_FOLDING_BI_DUAL"*
- *"MSK_CALLBACK_END_FOLDING_BI_INITIALIZE"*
- *"MSK_CALLBACK_END_FOLDING_BI_OPTIMIZER"*
- *"MSK_CALLBACK_END_FOLDING_BI_PRIMAL"*
- *"MSK_CALLBACK_END_INITIALIZE_BI"*
- *"MSK_CALLBACK_END_OPTIMIZE_BI"*
- *"MSK_CALLBACK_FOLDING_BI_DUAL"*
- *"MSK_CALLBACK_FOLDING_BI_OPTIMIZER"*
- *"MSK_CALLBACK_FOLDING_BI_PRIMAL"*
- *"MSK_CALLBACK_HEARTBEAT"*
- *"MSK_CALLBACK_OPTIMIZE_BI"*
- *"MSK_CALLBACK_QO_REFORMULATE"*
- *"MSK_DINF_FOLDING_BI_OPTIMIZE_TIME"*
- *"MSK_DINF_FOLDING_BI_UNFOLD_DUAL_TIME"*
- *"MSK_DINF_FOLDING_BI_UNFOLD_INITIALIZE_TIME"*
- *"MSK_DINF_FOLDING_BI_UNFOLD_PRIMAL_TIME"*
- *"MSK_DINF_FOLDING_BI_UNFOLD_TIME"*
- *"MSK_DINF_FOLDING_FACTOR"*
- *"MSK_DINF_FOLDING_TIME"*
- *"MSK_IINF_FOLDING_APPLIED"*
- *"MSK_IINF_MIO_FINAL_NUMBIN"*
- *"MSK_IINF_MIO_FINAL_NUMBINCONEVAR"*
- *"MSK_IINF_MIO_FINAL_NUMCON"*
- *"MSK_IINF_MIO_FINAL_NUMCONE"*
- *"MSK_IINF_MIO_FINAL_NUMCONEVAR"*
- *"MSK_IINF_MIO_FINAL_NUMCONT"*
- *"MSK_IINF_MIO_FINAL_NUMCONTCONEVAR"*

- *"MSK_IINF_MIO_FINAL_NUMDEXPCONES"*
- *"MSK_IINF_MIO_FINAL_NUMDJC"*
- *"MSK_IINF_MIO_FINAL_NUMDPOWCONES"*
- *"MSK_IINF_MIO_FINAL_NUMINT"*
- *"MSK_IINF_MIO_FINAL_NUMINTCONEVAR"*
- *"MSK_IINF_MIO_FINAL_NUMPEXPCONES"*
- *"MSK_IINF_MIO_FINAL_NUMPPOWCONES"*
- *"MSK_IINF_MIO_FINAL_NUMQCONES"*
- *"MSK_IINF_MIO_FINAL_NUMRQCONES"*
- *"MSK_IINF_MIO_FINAL_NUMVAR"*
- *"MSK_IINF_MIO_NUM_BLOCKS_SOLVED_IN_BB"*
- *"MSK_IINF_MIO_NUM_BLOCKS_SOLVED_IN_PRESOLVE"*
- *"MSK_LIINF_BI_CLEAN_ITER"*
- *"MSK_LIINF_FOLDING_BI_DUAL_ITER"*
- *"MSK_LIINF_FOLDING_BI_OPTIMIZER_ITER"*
- *"MSK_LIINF_FOLDING_BI_PRIMAL_ITER"*
- *"MSK_LIINF_MIO_FINAL_ANZ"*
- *"MSK_OPTIMIZER_NEW_DUAL_SIMPLEX"*
- *"MSK_OPTIMIZER_NEW_PRIMAL_SIMPLEX"*

Removed

- constant.callbackcode.begin_simplex_bi
- constant.callbackcode.im_bi
- constant.callbackcode.im_conic
- constant.callbackcode.im_dual_bi
- constant.callbackcode.im_intpnt
- constant.callbackcode.im_presolve
- constant.callbackcode.im_primal_bi
- constant.callbackcode.im_qo_reformulate
- constant.callbackcode.im_simplex_bi
- constant.dinfitem.bi_clean_dual_time
- constant.dinfitem.bi_clean_primal_time
- constant.liinfitem.bi_clean_dual_deg_iter
- constant.liinfitem.bi_clean_dual_iter
- constant.liinfitem.bi_clean_primal_deg_iter
- constant.liinfitem.bi_clean_primal_iter

Bibliography

[AA95] E. D. Andersen and K. D. Andersen. Presolving in linear programming. *Math. Programming*, 71(2):221–245, 1995.

[AGMeszarosX96] E. D. Andersen, J. Gondzio, Cs. Mészáros, and X. Xu. Implementation of interior point methods for large scale linear programming. In T. Terlaky, editor, *Interior-point methods of mathematical programming*, pages 189–252. Kluwer Academic Publishers, 1996.

[ART03] E. D. Andersen, C. Roos, and T. Terlaky. On implementing a primal-dual interior-point method for conic quadratic optimization. *Math. Programming*, February 2003.

[AY96] E. D. Andersen and Y. Ye. Combining interior-point and pivoting algorithms. *Management Sci.*, 42(12):1719–1731, December 1996.

[And09] Erling D. Andersen. The homogeneous and self-dual model and algorithm for linear optimization. Technical Report TR-1-2009, MOSEK ApS, 2009. URL: http://docs.mosek.com/whitepapers/homolo.pdf.

[And13] Erling D. Andersen. On formulating quadratic functions in optimization models. Technical Report TR-1-2013, MOSEK ApS, 2013. Last revised 23-feb-2016. URL: http://docs.mosek.com/whitepapers/qmodel.pdf.

[BKVH07] S. Boyd, S.J. Kim, L. Vandenberghe, and A. Hassibi. A Tutorial on Geometric Programming. *Optimization and Engineering*, 8(1):67–127, 2007. Available at http://www.stanford.edu/\protect\unhbox\voidb@x\penalty\@M\boyd/gp_tutorial.html.

[CCornuejolsZ14] M. Conforti, G. Cornu/'ejols, and G. Zambelli. *Integer programming*. Springer, 2014.

[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.

[Wol98] L. A. Wolsey. *Integer programming*. John Wiley and Sons, 1998.

Symbol Index

Enumerations

Functions

Parameters

Response codes

Index